Theoretical Computer Science 122 (1994) 3–47
Elsevier

A compositional semantics for logic programs

A. Bossi

Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, I-35131 Padova, Italy

M. Gabbrielli, G. Levi and M.C. Meo

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy

Abstract

Bossi, A., M. Gabbrielli, G. Levi and M.C. Meo, A compositional semantics for logic programs, Theoretical Computer Science 122 (1994) 3–47.

This paper considers *open logic programs* originally introduced as a tool to build an OR-compositional semantics of logic programs. We extend the original semantic definitions in the framework of the general approach to the semantics of logic programs described by Gabbrielli and Levi (1991). We first define an *operational semantics* $\mathcal{O}_\Omega(P)$ which models computed answer substitutions and which is compositional w.r.t. the union of programs. Next, we consider the semantic domain of *Ω-denotations*, which are sets of clauses with a suitable equivalence relation. The fixpoint semantics $\mathcal{F}_\Omega(P)$ given by Bossi and Menegus (1991) is proved equivalent to the operational semantics. From the model-theoretic viewpoint, an Ω-denotation is mapped onto a set of Herbrand interpretations, thus leading to a definition of an *Ω-model* based on the classical notion of truth. Moreover, we consider a particular kind of Ω-models (compositional models), and we show that $\mathcal{O}_\Omega(P)$ is a (nonminimal) compositional Ω-model. A suitable abstraction of $\mathcal{O}_\Omega(P)$ is compositional and fully abstract w.r.t. the equivalence induced by successful derivations. Finally, some applications of our semantics are discussed. In particular, $\mathcal{O}_\Omega(P)$ can be viewed as the foundation of modular program analysis.

1. Introduction

According to a popular view on logic programming, the problem of the semantics was solved once and for all by logicians before logic programming was even born.

Correspondence to: A Bossi, Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, I-35131 Padova, Italy. Email addresses of the authors: bossi@pdmat1.unipd.it, {gabbri, levi, meo}@di.unipi.it.

Namely, the only three important concepts are the *program* itself, the *intended interpretation* (declarative semantics) and the *theorem prover* (operational semantics). The program is a logic theory. The declarative semantics formalizes the application that the program is trying to capture. It is an interpretation in the conventional logic sense and a model of the program. Finally, the theorem prover is a proof procedure which must be sound and complete with respect to the declarative semantics. Is that really *all there is to it*?

The above view is appealing but too simple-minded to capture the difference between theorem proving and programming. In fact, it applies to any formal system for which there exists a sound and complete theorem prover. Theorem proving becomes logic programming when we restrict the class of theories, for example, to definite Horn clauses, so as to obtain a declarative semantics and a proof procedure similar to the denotational and the operational semantics of conventional programming languages. This is exactly what van Emden and Kowalski did in their seminal paper [55], where the proof procedure was SLD resolution and the representative model was the least Herbrand model. According to [55], *the* semantics is a mathematical object which is defined in model-theoretic terms and which can be *computed* both by a top-down and a bottom-up construction. Is the above classical and elegant result a satisfactory solution?

The answer can be found if we first consider a different and more basic question. What a semantics is used for? The first *application* of any semantics is to help understanding the meaning of programs. Other useful applications include areas such as program transformation and program analysis. One can argue that tens of thousands of logic programmers were really helped by the declarative understanding of their programs. One can also argue that semantics-based program transformation and analysis do require deeper results and more elaborate theories, but still only using basically the above-mentioned simple and straightforward semantics. The above arguments can become more technical only if we understand which is the basic semantic property of such formal activities as program transformation and analysis. The answer coming from computer science is *program equivalence*, i.e. program understanding is based on our ability to detect when two programs cannot be distinguished by looking at their behaviors.

Defining an equivalence on programs $\approx$ and a formal semantics $\mathscr{S}(P)$ are two strongly related tasks. A semantics $\mathscr{S}(P)$ is *correct* w.r.t. $\approx$, if $\mathscr{S}(P_1)=\mathscr{S}(P_2)$ implies $P_1 \approx P_2$. $\mathscr{S}(P)$ is *fully abstract* w.r.t. $\approx$, if the converse holds, i.e. if $P_1 \approx P_2$ implies $\mathscr{S}(P_1)=\mathscr{S}(P_2)$. While full abstraction is known to be a desirable property for any semantics, correctness is a must. The question on the adequacy of the van Emden and Kowalski's semantics can then be rephrased as follows. Is that semantics correct w.r.t. a "natural" notion of program equivalence? And this in turn raises the problem of choosing a satisfactory notion of program equivalence.

Equivalences can be defined by using logical arguments only. For example, one can use model-theoretic properties, such as the set of models, the set of logical consequences or the least Herbrand model, and proof-theoretic properties, such as the set of

derivable atoms. However, this would lead us nowhere, since the equivalence we need *must* be based on the operational behavior and on what we can observe from a computation. In other words, we have to learn more from computer science than from mathematical logic! From a computer science viewpoint, once we have a formalization of program execution, we have a choice for the equivalence. One important aspect of the formalization, in addition to the inference rules which specify how derivations are made, is the concept of *observable*, i.e. the property we observe in a computation. In logic programs we can be interested in different observable properties such as successful derivations, finite failures, computed answer substitutions, partial computed answer substitutions, etc. A given choice of the observable X induces an *observational equivalence* $\approx_X$ on programs. Namely, $P_1 \approx_X P_2$ iff P_1 and P_2 are observationally indistinguishable according to X. For example, if the observable s denotes *successful derivations*, $P_1 \approx_s P_2$ iff for any goal G, G is refutable in P_1 iff it is refutable in P_2. If the observable c denotes *computed answer substitutions*, $P_1 \approx_c P_2$ iff for any goal G, G has the same (up to renaming) computed answers in P_1 and in P_2.

Since the observable is the property which allows us to distinguish programs and is also the property we want to preserve in program transformations, the most natural choice in the case of logic programs is *computed answer substitutions*, which are exactly the result of a logic program execution. Other less abstract observables, such as partial computed answers and call patterns, might be useful in some applications, for example in semantics-based analysis and transformation. However, a more abstract observable like *successful derivations* would fail in capturing the essence of logic programming, even if it is the most adequate to the case of first-order theorem proving, where there is nothing to be returned as a result of the computation.

We can now note that, as first shown in [18, 19], *the van Emden and Kowalski's semantics is not correct w.r.t. the observational equivalence based on computed answer substitutions*, while it is correct w.r.t. the one based on successful derivations. Namely, there exist programs which have the same least Herbrand model and yet compute different answer substitutions. While trying to understand the meaning of programs, when analyzing and transforming programs, this semantics cannot be taken as the reference semantics. This is the reason why the need for a different formal semantics, correct w.r.t. answer substitutions, was recognized by many authors, giving rise to several new definitions (see e.g. [10, 21, 57, 18, 31]). The need for better semantics was also recognized in the case of semantics-based abstract interpretation [48, 37, 11, 3] and transformation [36, 5], where, as already mentioned, less abstract observables, such as partial computed answers or call patterns, have sometimes to be modeled.

The same problem was recognized even earlier in relation to some *extensions* of pure logic programming. In fact, the weakness of the traditional semantics is even more serious when trying to model the behavior of extensions such as constraint or concurrent logic programs, where the observables play a more important role. For example, finding definitions modeling the observable behavior was, from the very beginning, the aim of most of the research on the semantics of concurrent logic languages (see e.g. [43, 29, 24, 53, 13]). This was primarily motivated by the fact that

these languages closely resemble traditional concurrent languages, whose semantics usually model observables such as traces of input–output interactions, deadlocks, etc. However, one additional motivation was the fact that the "logical" declarative reading did not help the understanding of programs. Something similar took place in the case of constraint logic programming. In fact, even if the aim was not finding a semantics modeling the observable behavior, most of the constructions proposed by Jaffar and Lassez [34] are different from the van Emden and Kowalski construction (i.e. they are not $\mathfrak{R}$-models) and one of the proposed semantics was later proved [22] to correctly model the *answer constraint* observable.

It is worth noting that it is exactly the declarative reading of logic programs which fails in capturing the computed answers observable. In fact, even if we move from the least Herbrand model to the set of all the models, we find that there exist programs which have the same set of models and can still be distinguished by looking at their computed answers (and the converse holds too). Therefore, logical equivalence is not the same as the equivalence w.r.t. computed answers. The same statement is true for all the equivalence relations considered in [46].

In addition to the problem of modeling observational equivalences, there exists a very important property which does not hold in the traditional least Herbrand model semantics, i.e. *compositionality*. Compositionality has to do with a (syntactic) program composition operator $\circ$, and holds when the semantics of the compound construct $C_1 \circ C_2$ is defined by (semantically) composing the semantics of the constituents C_1 and C_2. In the case of logic programs, the construct which raises a compositionality problem is the *union* of clauses. The related property is sometimes called *OR-compositionality*. People got interested in OR-compositional semantics [39, 49, 47, 30, 31] both for theoretical and practical purposes, such as defining the semantics for modular versions of logic programs.

When compositions of programs is taken into account, for a given observable property we obtain different equivalences, depending on which kind of program composition we consider. Given an observable X and a program composition operator $\circ$, the induced congruence $\approx_{(\circ,X)}$ is defined as follows. $P_1 \approx_{(\circ,X)} P_2$ iff for any program Q, $P_1 \circ Q \approx_X P_2 \circ Q$, i.e. iff P_1 and P_2 are observationally indistinguishable under any possible context allowed by the composition operator. If the observable property is successful derivations, we find out that the least Herbrand model is not even OR-compositional. Still, OR-compositionality can be understood in logical terms. In fact, the set of all the models is correct w.r.t. successful derivations and OR-compositional [46].

Over the last few years we have developed a general approach to the semantics [23], whose aim was modeling the observable behaviors, possibly in a compositional way, for a variety of logic languages, ranging from positive logic programs [18–20, 7], to general logic programs [54, 27], constraint logic programs [22, 32] and concurrent constraint programs [24]. Our approach is based on the idea of choosing (equivalence classes of) sets of clauses as semantic domains. Our denotations are then defined by *syntactic* objects, as in the case of Herbrand interpretations. Denotations, that we

sometimes call π-interpretations, have some nice model-theoretic properties. However, they are not interpretations in the conventional mathematical logic sense, even if π-interpretations used in [18–20] can indeed be viewed as interpretations as done in [52, 38], where they were called *canonical realizations.* As in the case of the traditional van Emden and Kowalski semantics, our denotations can be computed both by a top-down construction (a success set) and by a bottom-up constructions (the least fixpoint of suitable continuous immediate consequence operators on π-interpretations). It is worth noting that our aim is not to define a new (artificial and futile) notion of model. We are simply unhappy with the traditional declarative semantics, because it characterizes the logical properties only, and we look for new notions of program denotation useful from the programming or computer science point of view. A satisfactory solution, even to the simple case of positive logic programs, is needed to gain a better understanding of more practical languages, such as real Prolog [2] and its purely declarative counterparts. A partial solution was the *s*-semantics [18, 19], which was the first (noncompositional) semantics correct w.r.t. computed answers and which used sets of unit clauses as semantic domain.

In this paper we extend the *s*-semantics approach to compositionality, starting from the results in [7]. Note that all the existing OR-compositional semantics (apart from those in [31, 7]) are correct w.r.t. successful derivations only [39, 49, 47, 46, 30]. Gaifman and Shapiro [30] first introduced an OR-compositional semantics in a proof-theoretic framework. OR-compositionality is achieved by using sets of nonunit clauses as semantic domain. This semantics was then extended to model computed answers in [31]. The resulting denotation, which is also fully abstract, uses semantic domains more complex than sets of clauses. However, the main reason why we are not happy with the above semantics is that we want to characterize our denotations according to the van Emden and Kowalski's style, namely by a top-down and a bottom-up computation process. These characterizations are in fact useful to effectively compute approximations of the denotation, as in the case of abstract interpretation. The Ω-semantics [7] was the first real compositional generalization of the *s*-semantics and was defined by a fixpoint construction. This semantics is based on the notion of an Ω-open program. An *Ω-open program* P is a program in which the predicate symbols belonging to the set Ω are considered partially defined in P. P can be composed with other programs which may further specify the predicates in Ω. Such a composition is a generalization of program union and is formally defined later (Definition 2.3). A typical partially defined program is a program where the intensional definitions are completely known while extensional definitions are only partially known and can be further specified.

Example 1.1. Let us consider the following program:

$$
\begin{aligned}
Q_1 = \{ & \mathit{ancestor}(X, Y) \text{:-} \mathit{parent}(X, Y). \\
& \mathit{ancestor}(X, Z) \text{:-} \mathit{parent}(X, Y), \mathit{ancestor}(Y, Z). \\
& \mathit{parent}(\mathit{isaac}, \mathit{jacob}). \\
& \mathit{parent}(\mathit{jacob}, \mathit{benjamin}). \qquad \}.
\end{aligned}
$$

New extensional information defining new parent tuples can be added to Q_1 as follows:

$$Q_2 = \{\, parent(anna, elizabeth).$$
$$\quad parent(elizabeth, john).\,\}.$$

The semantics of open programs must be compositional w.r.t. program union, i.e. the semantics of $P_1 \cup P_2$ must be derivable from the semantics of P_1 and P_2.

As already noted, the least Herbrand model semantics, as originally proposed in [55], and the computed answer substitution semantics in [18–20], are not compositional w.r.t. program union. For example, in Example 1.1, the atom *ancestor*(*anna*, *elizabeth*), which belongs to the least Herbrand model of $Q_1 \cup Q_2$, cannot be obtained from the least Herbrand model of Q_1 and Q_2 (see also Example 2.1). Note that here we are interested in a composition of programs which *extends* the definition of a predicate already existing, and we do not consider any *overriding* mechanisms. Of course, such mechanisms are also interesting. For example, when considering programs structured by using inheritance operators, in some cases we want the new information to replace the old one. The specific semantic treatment for inheritance by extension and/or overriding is described in [4], and is obtained by a modification of the framework we show here.

In this paper we generalize the semantics given in [7] for Ω-open programs, following the general approach in [23]. In the specific case of Ω-open programs, π-interpretations are called Ω-denotations and are sets of *conditional atoms*, i.e., clauses such that all the atoms in the body are open. We give a fixpoint semantics, which is a variation of the one proposed in [7], described in terms of Ω-denotations and an equivalent operational semantics. The resulting denotation is then characterized from the model-theoretic viewpoint, by defining a set of denotations which encompass standard Herbrand models. Particular denotations are called Ω-models, and are based on the standard notion of truth and on the fact that each Ω-denotation represents the set of Herbrand interpretations that can be obtained by composing the open program with a definition for the open predicates. Ω-models of open programs are introduced to obtain a unique representative denotation, computable as the least fixpoint of a suitable continuous operator, in cases where no such representative exists in the set of Herbrand models. Our Ω-models are related to the S_Ω-models defined in [7] by means of an ad hoc notion of truth.

The rest of this paper is organized as follows. Section 1.1 contains notation and useful definitions on the semantics of logic programs. In Section 2 we define an *operational semantics* $\mathcal{O}_\Omega(P)$ modeling computed answer substitutions which is *OR-compositional*. Section 3 introduces a suitable *semantic domain* for the $\mathcal{O}_\Omega(P)$ semantics and defines *Ω-denotations* which are sets of clauses modulo a suitable equivalence relation. In Section 4 the *fixpoint semantics* $\mathcal{F}_\Omega(P)$ is proved equivalent to the operational semantics. Section 5 is concerned with *model theory*. From the model-theoretic viewpoint, an Ω-denotation is mapped onto a set of Herbrand interpretations, thus leading to a definition of an Ω-model based on the classical notion of truth.

Moreover, we consider a particular kind of Ω-models (compositional models). $\mathscr{F}_\Omega(P)$ is a (nonminimal) compositional Ω-model equivalent to the model-theoretic semantics defined in [7] in terms of S_Ω-models. A suitable abstraction of $\mathscr{O}_\Omega(P)$ is a compositional Ω-model which is fully abstract w.r.t. the equivalence induced by successful derivations. A comparison between Ω-models and the S_Ω-models is made in Section 6. Finally, Section 7 discusses some applications of our semantics and points out some connections with other works. Some proofs are deferred to the appendix. A preliminary short version of this paper appeared in [6].

1.1. Preliminaries

The reader is assumed to be familiar with the terminology of and the basic results in the semantics of logic programs [44, 1]. Let the signature S consist of a set F of *function symbols*, a finite set Π of *predicate symbols*, a denumerable set V of *variable symbols*. All the definitions in the following will assume a given signature S. Let T be the set of terms built on F and V. Variable-free terms are called *ground* terms. A substitution is a mapping $\vartheta: V \to T$ such that the set $D(\vartheta) = \{X \mid \vartheta(X) \neq X\}$ (*domain* of ϑ) is finite. If $W \subset V$, we denote by $\vartheta|_W$ the *restriction* of ϑ to the variables in W, i.e. $\vartheta|_W(Y) = Y$ for $Y \notin W$. ε denotes the empty substitution. The *composition* $\vartheta\sigma$ of the substitutions ϑ and σ is defined as the functional composition. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \varepsilon$. The preordering $\leqslant$ (more general than) on substitutions is such that $\vartheta \leqslant \sigma$ iff there exists ϑ' such that $\vartheta\vartheta' = \sigma$. The result of the application of the substitution ϑ to a term t is an *instance* of t denoted by $t\vartheta$. We define $t \leqslant t'$ (t is more general than t') iff there exists ϑ such that $t\vartheta = t'$. A substitution ϑ is *grounding* for t if $t\vartheta$ is ground. The relation $\leqslant$ is a preorder. $\sim$ denotes the associated equivalence relation (*variance*). A substitution ϑ is a *unifier* of terms t and t' if $t\vartheta \equiv t'\vartheta$. It is well known that if two terms are unifiable then they have an idempotent most general unifier which is unique up to renaming. Therefore, $mgu(t_1, t_2)$ will denote such a most general unifier of t_1 and t_2. All the above definitions can be extended to other syntactic expressions in the obvious way. An atom is an object of the form $p(t_1, \ldots, t_n)$, where $p \in \Pi$, $t_1, \ldots, t_n \in T$. A *clause* is a formula of the form $H :- L_1, \ldots, L_n$ with $n \geqslant 0$, where H (the *head*) and $L_1, \ldots, L_n$ (the *body*) are atoms. ":-" and "," denote logic implication and conjunction, respectively, and all variables are universally quantified. If the body is empty, the clause is a *unit clause*. A clause $H :- B_1, \ldots, B_n$ subsumes the clause $K :- D_1, \ldots, D_m$ iff there exists a substitution ϑ such that $H\vartheta = K$ and $\{B_1\vartheta, \ldots, B_n\vartheta\} \subseteq \{D_1, \ldots, D_m\}$. A *program* is a finite set of clauses. A *goal* is a formula $L_1, \ldots, L_m$, where each L_i is an atom. By $Var(E)$ and $Pred(E)$ we denote, respectively, the sets of variables and predicates occurring in the expression E. A *Herbrand interpretation* I for a program P is a set of ground atoms. The intersection $M(P)$ of all the Herbrand models of a program P is a model (least Herbrand model). $M(P)$ is also the least fixpoint of a continuous transformation T_P (*immediate consequences operator*) on the complete lattice of Herbrand interpretations. If G is a goal, $G \overset{\vartheta}{\rightsquigarrow}_{P,R} B_1, \ldots, B_n$ denotes an SLD derivation

of the resolvent $B_1, \ldots, B_n$ from G in the program P which uses the selection rule R and where ϑ is the composition of the mgu's used in the derivation. If R is omitted, we mean that any selection rule is used (and the definition is independent from the selection rule). $\square$ denotes the empty clause, therefore $G \overset{\vartheta}{\rightsquigarrow} \square$ denotes the refutation of G in the program P. The computed answer substitution of a refutation $G \overset{\vartheta}{\rightsquigarrow} \square$ is the substitution obtained by the restriction of ϑ to the variables occurring in G. $G \overset{\vartheta}{\mapsto} \square$ will denote explicitly the refutation of G in P with computed answer substitution β. The notations $\tilde{t}$ and $\tilde{X}$ will be used to denote tuples of terms and variables, respectively, while $\tilde{B}$ denotes a (possibly empty) conjunction of atoms.

2. Computed answer substitution semantics for Ω-open programs

The operational semantics of logic programs with computed answer substitutions as observable [18–20] can be defined as

$$\mathcal{O}(P) = \{ p(X_1, \ldots, X_n)\vartheta \mid p(X_1, \ldots, X_n) \overset{\vartheta}{\mapsto}_P \square \}.$$

The denotation of a program is a set of nonground atoms, which can be viewed as a possibly infinite program. More precisely, a denotation is a (possibly infinite) set of equivalence classes of clauses, or a π-interpretation according to [23]. The equivalence is needed to abstract from irrelevant syntactic differences and in the above semantics it is simply the variance relation.

The denotation of a program P is a π-interpretation I, which has the following property. P and I are observationally equivalent with respect to any goal G. This is the property which allows us to state that the semantics does indeed capture the observable behavior [19]. The above semantics (as well as the least Herbrand model semantics), however, is only compositional w.r.t. the conjunction of atoms in a goal or in a clause body. In fact, the denotation of a conjunction can be obtained from the denotation of its conjuncts. The following example shows that when considering OR-composition (i.e. union of sets of clauses), nonground atoms (or unit clauses) are no longer sufficient to define a compositional semantics.

Example 2.1. Let us consider the following programs:

$$P_1 = \{ q(X) :- p(X). \quad r(X) :- s(X). \quad s(b). \quad p(a). \quad \}. \qquad P_2 = \{ p(b). \}$$

According to the previous definition of $\mathcal{O}(P)$, $\mathcal{O}(P_1) = \{ p(a), q(a), r(b), s(b) \}$ and $\mathcal{O}(P_2) = \{ p(b) \}$. Since $\mathcal{O}(P_1 \cup P_2) = \{ p(a), p(b), q(a), q(b), r(b), s(b) \}$, the semantics of

the union of the two programs cannot be obtained from the semantics of the programs. Note that also the least Herbrand model is not compositional w.r.t. $\cup$ since, for previous programs P_1, P_2 and $P_1 \cup P_2$, $M(P)$ is the same as $\mathcal{O}(P)$.

In order for a semantics to be compositional it must contain information in the form of a mapping from sets of atoms to sets of atoms. This is indeed the case of the semantics based on the closure operator [39] and on the T_P operator [49, 47]. If we want a semantics expressed by the program syntax, compositionality w.r.t. union of programs can only be obtained by choosing as semantic domain a set of (equivalence classes of) clauses. In Example 2.1, for instance, the semantics of P_1 should also contain the clause $q(X) \text{:-} p(X)$. Let us formally give the definition of the program composition we consider.

Definition 2.2 (*Ω-open program*). An Ω-open program (Ω-program for short) is a logic program P together with a set Ω of predicate symbols. A predicate symbol occurring in Ω is considered to be only partially defined in P.

Definition 2.3 (*Ω-union*). Let P_1 be an Ω_1-program and P_2 be an Ω_2-program. If

(1) $\Omega \subseteq \Omega_1 \cup \Omega_2$ and

(2) $(Pred(P_1) \cap Pred(P_2)) \subseteq (\Omega_1 \cap \Omega_2)$,

then $P_1 \cup_\Omega P_2$ is the Ω-open program $P_1 \cup P_2$. Otherwise, $P_1 \cup_\Omega P_2$ is not defined.

The definition of any predicate symbol $p \in \Omega$ in an Ω-open program P can always be extended or refined. For instance, in Example 1.1 program Q_1 is open w.r.t. the predicate *parent* and this predicate is refined in program Q_2. Therefore, a deduction concerned with a predicate symbol of an Ω-open program P can be either *complete* (when it takes place completely in the program P) or *partial* (when it terminates in P with an atom $p(\tilde{t})$ such that $p \in \Omega$ and $p(\tilde{t})$ does not unify with the head of any clause in P). A partial deduction can be completed by the addition of new clauses. Thus, we have a *hypothetic deduction*, conditional on the extension of predicate p.

Let us consider again the program P_1 of Example 2.1 and assume $\Omega = \{p\}$. Then, the goal $r(X)$ produces a complete deduction only, computing the answer substitution $\{X/b\}$. The goal $q(X)$ produces a complete deduction computing the answer substitution $\{X/a\}$, and a hypothetical deduction returning any answer that could be computed by a definition of p external to P_1. The goal $q(b)$ has one hypothetical deduction only, conditional on the provability (outside P_1) of $p(b)$. We want to express this hypothetical reasoning, i.e. that $q(b)$ is refutable if $p(b)$ is refutable. Hence, we will consider the following operational semantics.

Definition 2.4. Let Ω be a set of predicate symbols. We define

$$Id_\Omega = \{ p(X_1, \ldots, X_n) \text{:-} p(X_1, \ldots, X_n) \mid p \in \Omega, \text{ the arity of } p \text{ is } n \text{ and } X_1, \ldots, X_n \text{ are distinct variables} \}.$$

Definition 2.5 (*Ω-compositional operational semantics*). Let P be a program and Ω a set of predicate symbols. Moreover, let $P^* = P \cup Id_\Omega$ and let R be a fair selection rule. Then we define

$$\mathcal{O}_\Omega(P) = \{ A :- B_1, \ldots, B_n \mid p(X_1, \ldots, X_k) \overset{\vartheta}{\rightsquigarrow}_{P,R} A_1, \ldots, A_m \overset{\gamma}{\rightsquigarrow}_{P^*,R} B_1, \ldots, B_n, \; X_1, \ldots, X_k \text{ are distinct variables, } A = p(X_1, \ldots, X_k)\vartheta\gamma \text{ and } Pred(B_1, \ldots, B_n) \subseteq \Omega \}.$$

Moreover, if P is an Ω-open program, $\mathcal{O}_\Omega(P)$ is also Ω-open.

Note that, for $\Omega = \emptyset$, i.e. when all the predicates are considered completely defined, the previous definition boils down to the definition of the *s*-semantics [18]. $\mathcal{O}_\Omega(P)$ is a set of *resultants* [45] obtained from goals of the form $p(\tilde{X})$ in P and is essentially the result of the partial evaluation of P, where derivations terminate at open predicates (i.e. predicates in Ω). The set of clauses Id_Ω in the previous definition is used to delay the evaluation of open atoms. As shown by Proposition 2.6, this is a trick which allows one to obtain, by using a fixed fair selection rule R, all the derivations $p(X_1, \ldots, X_k) \overset{\vartheta}{\rightsquigarrow}_{P,R'} B_1, \ldots, B_n$ which use any selection rule R' for $Pred(B_1, \ldots, B_n) \subseteq \Omega$. Therefore, the previous definition is independent from the fair selection rule considered. Note that in the first step of the derivations we use clauses from P (instead of from P^*) because we want $\mathcal{O}_\Omega(P)$ to contain a clause $p(\tilde{X}) :- p(\tilde{X})$ if and only if $p(\tilde{X}) \overset{\varepsilon}{\rightsquigarrow}_P p(\tilde{X})$. The proof of the following proposition is in the appendix.

Proposition 2.6. *Let R be a fair selection rule, let $P^* = P \cup Id_\Omega$, $\tilde{X}$ a tuple of distinct variables and $Pred(B_1, \ldots, B_n) \subseteq \Omega$. Then there exists a rule R' such that $p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{P,R'} B_1, \ldots, B_n$ iff $p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P,R} D_1, \ldots, D_m \overset{\sigma}{\rightsquigarrow}_{P^*,R} B_1, \ldots, B_n$ and $p(\tilde{X})\gamma\sigma = p(\tilde{X})\vartheta$.*

Example 2.7. Let P_1, P_2 be the Ω-open programs of Example 2.1, where $\Omega = \{p\}$. Then

$$\mathcal{O}_\Omega(P_1) = \{ p(a), q(a), r(b), s(b), q(X) :- p(X) \},$$

$$\mathcal{O}_\Omega(P_2) = \{ p(b) \}.$$

$\mathcal{O}_\Omega$ contains enough information to compute the semantics of the composition. In fact,

$$\mathcal{O}(P_1 \cup P_2) \subseteq \mathcal{O}_\Omega(P_1 \cup P_2) = \{ p(a), p(b), q(a), q(b), r(b), s(b), q(X) :- p(X) \}$$

and

$$\mathcal{O}_\Omega(P_1 \cup P_2) = \mathcal{O}_\Omega(\mathcal{O}_\Omega(P_1) \cup \mathcal{O}_\Omega(P_2))$$

(see Theorem 2.13).

The congruence ($\approx_\Omega$) on programs induced by the computed answer substitution observable when considering also the programs union, can formally be defined as follows.

Definition 2.8. Let P_1, P_2 be Ω-open programs. Then $P_1 \approx_\Omega P_2$ if for every goal G and for every Ω-program Q s.t. $P_i \cup_\Omega Q$, $i=1,2$, is defined, $G \overset{\vartheta_1}{\rightsquigarrow}_{P_1 \cup_\Omega Q} \square$ iff $G \overset{\vartheta_2}{\rightsquigarrow}_{P_2 \cup_\Omega Q} \square$ and $G\vartheta_1 = G\vartheta_2$ up to renaming.

$\mathcal{O}_\Omega$ allows one to characterize a notion of answer substitution which enhances the usual one, since (unresolved) atoms, with predicate symbols in Ω, are also considered. Therefore, it is able to model computed answer substitutions in an OR-compositional way. Indeed, Theorem 2.10 shows that a program P and its operational semantics $\mathcal{O}_\Omega(P)$ are $\approx_\Omega$ equivalent. As a consequence, the semantics $\mathcal{O}_\Omega(P)$ correctly captures the computed answer substitution observable when considering also the programs union, i.e. $\mathcal{O}_\Omega(P)$ is correct w.r.t. the equivalence $\approx_\Omega$ (Corollary 2.11). Theorem 2.13 shows the compositionality of the semantics w.r.t. the operator $\cup_\Omega$. The proof of Lemma 2.12 is also in the appendix.

The proof of Lemma 2.9 is in the appendix and is an extension of the proof of the completeness Theorem in [19].

Lemma 2.9. *Let P be a program. Then $p(\tilde{t}) \overset{\vartheta_1}{\rightsquigarrow}_{P,R} G_1$ iff*

- $p(\tilde{X}) \overset{\vartheta_2}{\rightsquigarrow}_{P,R} G_2$ *and*
- *there exists $\sigma = mgu(p(\tilde{t}), p(\tilde{X})\vartheta_2)$, $p(\tilde{t})\vartheta_1 = p(\tilde{X})\vartheta_2\sigma$ and $G_2\sigma = G_1$.*

Theorem 2.10. *Let P be an Ω-open program. Then $P \approx_\Omega \mathcal{O}_\Omega(P)$.*

Proof. We have to show that for every Ω-open program Q such that $\mathcal{O}_\Omega(P) \cup_\Omega Q$ and $P \cup_\Omega Q$ are defined, $G \overset{\gamma_2}{\rightsquigarrow}_{\mathcal{O}_\Omega(P) \cup Q} \square$ iff $G \overset{\gamma_1}{\rightsquigarrow}_{P \cup Q} \square$, where $G\gamma_2 = G\gamma_1$ (up to renaming) (note that if $P \cup_\Omega Q$ is defined then $\mathcal{O}_\Omega(P) \cup_\Omega Q$ is also defined but the converse is not true). By the independence from the selection rule for SLD refutations [44, 1], we can assume that the selection of the atoms is performed according to the following rule denoted by $\mathcal{S}$:

(1) first select the nonopen atoms (i.e. the $p(\tilde{t})$'s such that $p \notin \Omega$),

(2) among the nonopen atoms, first select those which are added to the current resolvent by the last inference step (i.e. those in the body of the last used clause).

We will show that, when considering successful derivations, $G \overset{\vartheta_2}{\rightsquigarrow}_{\mathcal{O}_\Omega(P) \cup Q, \mathcal{S}} R$ in one step iff there exists n such that $G \overset{\vartheta_1}{\rightsquigarrow}_{P \cup Q, \mathcal{S}} R$ in n steps and $G\vartheta_1 = G\vartheta_2$ (up to renaming). The thesis follows from the above result by a straightforward inductive argument and by definition of $\approx_\Omega$. In the following we consider resolvents as multisets, and we denote by $(G \setminus A), B$ the multiset obtained from the multiset of atoms G by deleting the atom A and by adding the multiset of atoms B. Let $p(\tilde{t})$ be the atom selected in G. If $p(\tilde{t})$ is reduced by using a clause in Q, the thesis follows with $n=1$. Otherwise let R be defined as follows:

(1) $(R = G \setminus p(\tilde{t}))\vartheta_1, B$ is the first resolvent ($\neq G$) in the derivation $G \overset{\gamma_1}{\rightsquigarrow}_{P \cup Q, \mathcal{S}} \square$ such that $\mathcal{S}(R) = A\vartheta_1$ and either $Pred(A) \in \Omega$ or $A \in G$. Note that by definition of $\mathcal{S}$, $Pred(B) \subseteq \Omega$.

(2) If there does not exist an R as specified in (1), then R is the empty resolvent. Such an R exists since we are considering finite (successful) derivations. For R as specified in (1), we have

$$
\begin{aligned}
&G \overset{\vartheta_1}{\rightsquigarrow}_{P\cup Q,\mathscr{S}} R && \text{iff (by definition of } R \text{ and of } \mathscr{S})\\
&p(\tilde{t}) \overset{\vartheta_1}{\rightsquigarrow}_{P,\mathscr{S}} B && \text{iff (by Lemma 2.9)}\\
&p(\tilde{X}) \overset{\vartheta_2}{\rightsquigarrow}_{P,\mathscr{S}} B_2 && \text{iff (by Definition 2.5 and Proposition 2.6)}\\
&p(\tilde{X})\vartheta_2 \text{ :- } B_2 \in \mathscr{O}_\Omega(P) && \text{iff (by definition of } \rightsquigarrow_{\mathscr{O}_\Omega(P),\mathscr{S}})\\
&G \overset{\sigma}{\rightsquigarrow}_{\mathscr{O}_\Omega(P),\mathscr{S}} (G\backslash p(\tilde{t}))\sigma, B_2\sigma,
\end{aligned}
$$

where $\sigma = mgu(p(\tilde{X})\vartheta_2, p(\tilde{t}))$, $G\sigma = G\vartheta_1$ and $B_2\sigma = B$ (by Lemma 2.9). For R as in (2) the same holds with R, B, B_2 and $G\backslash p(\tilde{t}) = \emptyset$ and this completes the proof. □

Corollary 2.11 (Correctness). *Let P_1, P_2 be Ω-open programs. If $\mathscr{O}_\Omega(P_1) = \mathscr{O}_\Omega(P_2)$ then $P_1 \approx_\Omega P_2$.*

Proof. Straightforward by Theorem 2.10. □

Lemma 2.12. *Let P be a program and G be a goal. Then $G \overset{\vartheta}{\rightsquigarrow}_{P,R} N$ iff $G \overset{\vartheta'}{\rightsquigarrow}_{P,R'} N$, where $\vartheta|_{Var(G)} = \vartheta'|_{Var(G)}$ and the derivation $G \overset{\vartheta'}{\rightsquigarrow}_{P,R'} N$ is obtained from $G \overset{\vartheta}{\rightsquigarrow}_{P,R} N$ by changing the order in which the atoms are selected.*

Theorem 2.13 (Compositionality). *Let P_1 be an Ω_1-open program, P_2 an Ω_2-open program and let $P_1 \cup_\Omega P_2$ be defined. Then $\mathscr{O}_\Omega(\mathscr{O}_{\Omega_1}(P_1) \cup_\Omega \mathscr{O}_{\Omega_2}(P_2)) = \mathscr{O}_\Omega(P_1 \cup_\Omega P_2)$.*

Proof. First note that, by Definition 2.5, $Pred(\mathscr{O}_\Omega(P)) \subseteq Pred(P)$. Therefore, by Definition 2.3, if $P_1 \cup_\Omega P_2$ is defined then $\mathscr{O}_{\Omega_1}(P_1) \cup_\Omega \mathscr{O}_{\Omega_2}(P_2)$ is also defined. Moreover, recall that if $P_1 \cup_\Omega P_2$ is defined, then $P_1 \cup_\Omega P_2$ is the Ω-open program $P_1 \cup P_2$. By Definition 2.5 and by Proposition 2.6, it is then sufficient to show that $\exists R$ such that

$$p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{P_1\cup P_2,R} B_1, \ldots, B_n$$

iff $\exists R'$ such that

$$p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{\mathscr{O}_{\Omega_1}(P_1)\cup\mathscr{O}_{\Omega_2}(P_2),R'} B_1, \ldots, B_n$$

with $Pred(B_1, \ldots, B_2) \subseteq \Omega$.

Let us prove the two implications separately.

"If": Assume, without loss of generality, that

$$A_1, \ldots, A_{i-1}, p(\tilde{t}), A_{i+1}, \ldots, A_m \overset{\vartheta}{\rightsquigarrow}_{\mathscr{C}_{\Omega_1}(P_1), R'} (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\vartheta$$

in one step, using the clause $p(\tilde{l}) :- B_1, \ldots, B_n \in \mathscr{C}_{\Omega_1}(P_1)$ with $\vartheta = mgu(p(\tilde{t}), p(\tilde{l}))$. By Definition 2.5 and by Proposition 2.6, $\exists R$ such that $p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P_1, R} B_1, \ldots, B_n$ with $p(\tilde{X})\gamma = p(\tilde{l})$ (and hence $\vartheta = mgu(p(\tilde{X})\gamma, p(\tilde{t}))$). Then, by Lemma 2.9, $p(\tilde{t}) \overset{\gamma_1}{\rightsquigarrow}_{P_1, R} B'_1, \ldots, B'_n$ with $p(\tilde{t})\gamma_1 = p(\tilde{X})\gamma\vartheta$ and $B'_1, \ldots, B'_n = (B_1, \ldots, B_n)\vartheta$. Hence, in P_1 there exists the derivation

$$A_1, \ldots, A_{i-1}, p(\tilde{t}), A_{i+1}, \ldots, A_m \overset{\gamma_1}{\rightsquigarrow}_{P_1, R}$$

$$(A_1, \ldots, A_{i-1})\gamma_1, B'_1, \ldots, B'_n, (A_{i+1}, \ldots, A_m)\gamma_1.$$

By definition of $B'_1, \ldots, B'_n$ and since the bindings for variables in $A_1, \ldots, A_m$ are determined by the variables in $p(\tilde{t})$, we have

$$(A_1, \ldots, A_{i-1}, p(\tilde{t}), A_{i+1}, \ldots, A_m)\gamma_1 = (A_1, \ldots, A_{i-1}, p(\tilde{t}), A_{i+1}, \ldots, A_m)\vartheta$$

and

$$(A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\vartheta =$$

$$(A_1, \ldots, A_{i-1})\gamma_1, B'_1, \ldots, B'_n (A_{i+1}, \ldots, A_m)\gamma_1.$$

Therefore, the thesis holds by a straightforward inductive argument.

Only if: Assume that $G \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup P_2, R} B_1, \ldots, B_n$ with $Pred(B_1, \ldots, B_n) \subseteq \Omega$, and, without loss of generality, assume that the first atom selected is $p(\tilde{t})$ and the first clause used in the derivation is in P_1. Since $Pred(B_1, \ldots, B_n) \subseteq \Omega$, by Lemma 2.12 we can assume that the selection rule R is $\mathscr{S}$ as specified in the proof of Theorem 2.10, considering as nonopen atoms the $p(\tilde{t})$'s such that $p \notin \Omega_1$. Also, the notation $(G \setminus A), B$ is the same as in such a proof. Let N be defined as follows:

(1) $N = (G \setminus p(\tilde{t}))\vartheta_1, B$ is the first resolvent ($\neq G$) in the derivation

$$G \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup P_2, \mathscr{S}} B_1, \ldots, B_n$$

such that $\mathscr{S}(N) = A\vartheta_1$ and either $Pred(A) \in \Omega_1$ or A is an atom in G,

(2) $N = B_1, \ldots, B_n$ if there does not exist any N as specified in (1).

Note that, by definition of $\cup_\Omega$, if $P_1 \cup_\Omega P_2$ is defined, then $Pred(P_1) \cap Pred(P_2) \subseteq (\Omega_1 \cap \Omega_2) \subseteq \Omega_1$. Therefore, every atom A selected before N is rewritten using a clause in P_1. Moreover, note that, by definition of $\mathscr{S}$, $Pred(N) \subseteq \Omega_1$. Therefore, we can repeat the argument of the proof of Theorem 2.10 to show that

$$G \overset{\vartheta_1}{\rightsquigarrow}_{P_1, \mathscr{S}} N$$

iff

$$G \overset{\sigma}{\rightsquigarrow}_{\mathcal{O}_{\Omega_1}(P_1),\mathcal{S}} N,$$

where $G\sigma = G\vartheta_1$. Then we have the following implications:

$$G \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup P_2,\mathcal{S}} B_1, \ldots, B_n \qquad \text{iff (by definition of } N \text{ and of } \mathcal{S})$$

$$G \overset{\vartheta_1}{\rightsquigarrow}_{P_1,\mathcal{S}} N \overset{\gamma}{\rightsquigarrow}_{P_1 \cup P_2,\mathcal{S}} B_1, \ldots, B_n \qquad \text{iff (by previous iff)}$$

$$G \overset{\vartheta_1}{\rightsquigarrow}_{\mathcal{O}_{\Omega_1}(P_1),\mathcal{S}} N \overset{\gamma}{\rightsquigarrow}_{P_1 \cup P_2,\mathcal{S}} B_1, \ldots, B_n,$$

where $G\vartheta_1\gamma = G\vartheta$. Therefore, by symmetry and a straightforward inductive argument, we have that $p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup P_2,\mathcal{S}} B_1, \ldots, B_n$ iff $p(\tilde{X}) \overset{\beta}{\rightsquigarrow}_{\mathcal{O}_{\Omega_1}(P_1) \cup \mathcal{O}_{\Omega_2}(P_2),\mathcal{S}} B_1, \ldots, B_n$ with $p(\tilde{X})\beta = p(\tilde{X})\vartheta$, $Pred(B_1, \ldots, B_n) \subseteq \Omega$ and this completes the proof. □

Note that when considering $\Omega = \emptyset$, Theorem 2.18 is the completeness theorem of the s-semantics in [18].

3. Semantic domain for Ω-open programs

In this section we formally define the semantic domain which characterizes the above-introduced operational semantics $\mathcal{O}_\Omega$. Since $\mathcal{O}_\Omega$ contains clauses (whose body predicates are all in Ω), we have to accommodate clauses in the semantic domain we use. The π-interpretations of Ω-open programs will be called *Ω-denotations*. An Ω-denotation can be viewed as a function from interpretations to interpretations since it contains conditional atoms. As usual, in the following, Ω is a set of predicates.

Definition 3.1 (*Conditional atoms*). An Ω-conditional atom is a clause $A :- B_1, \ldots, B_n$ such that $Pred(B_1, \ldots, B_n) \subseteq \Omega$.

In order to abstract from the purely syntactical details, we have to use an equivalence on conditional atoms. We use variance, considering bodies as multisets. Coarser definitions (correct w.r.t. $\approx_\Omega$) can be used. $\equiv$ denotes syntactic identity.

Definition 3.2. Let $c_1 = A_1 :- B_1, \ldots, B_n$, $c_2 = A_2 :- D_1, \ldots, D_m$ be two clauses. Then $c_1 \leqslant_c c_2$ iff there exists a substitution σ and there exists $\{i_1, \ldots, i_n\} \subseteq \{1, \ldots, m\}$ such that $(A_1 :- B_1, \ldots, B_n)\sigma \equiv A_2 :- D_{i_1}, \ldots, D_{i_n}$. We denote by $\simeq$ the equivalence induced by $\leqslant_c$. Moreover, we denote still by $\simeq$ the equivalence on sets of clauses defined as follows. $P_1 \simeq P_2$ iff $\forall c_2 \in P_2 \exists c_1 \in P_1$ such that $c_1 \simeq c_2$ and vice versa.

It is straightforward to show that $\leqslant_c$ is a preorder and therefore $\simeq$ is an equivalence. The extension of $\leqslant_c$ to $\simeq$-equivalence classes is then an ordering, and in the following will still be denoted by $\leqslant_c$. Note that, by the previous definition, if $c_1 \simeq c_2$ then there exists a renaming ρ such that the clauses $c_1\rho$ and c_2 have the same heads and the same bodies, by considering bodies as multisets of atoms (instead of sequences). The following is an example of the previous definitions.

Example 3.3. Let us consider the clauses
c_1 $p(X, Y) :- q(W, Y)$,
c_2 $p(X, Y) :- q(W, Y), q(Z, Y)$,
c_3 $p(X, b) :- q(a, b), r(X, Y)$.
Then $c_1 \leqslant_c c_3$, while $c_2 \not\leqslant_c c_3$. Note that both c_1 and c_2 subsume c_3.

Note also that considering bodies of clauses as sets instead of multisets in the definition of $\leqslant_c$ (i.e. considering subsumption equivalence as $\simeq$) would not be correct w.r.t. computed answer substitutions. The following is a simple counterexample.

Example 3.4. Let us consider the clauses
c_1 $p(X, Y) :- q(X, Y), q(X, Y)$,
c_2 $p(X, Y) :- q(X, Y)$.
Let $P_1 = \{c_1\}$ and $P_2 = \{c_2\}$ be Ω-open programs with $\Omega = \{q\}$. If bodies of clauses are viewed as sets, c_1 and c_2 could be considered equivalent (obviously c_1 subsumes c_2 and vice versa). However, $P_1 \not\approx_\Omega P_2$. In fact, let Q be the program $\{q(X, b), q(a, Y)\}$. Then $p(X, Y) \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup Q} \square$ where $\vartheta = \{X/a, Y/b\}$, while the goal $p(X, Y)$ in the program $P_2 \cup Q$ can compute either $\{X/a\}$ or $\{Y/b\}$ only. Note that $c_1 \not\simeq c_2$.

Obviously $\simeq$ is finer than $\approx_\Omega$, i.e. $\simeq$ equivalent clauses cannot be distinguished in any context w.r.t. the computed answer substitutions semantics. As a consequence, we can use $\simeq$ in the definition of the semantic domain as follows.

Definition 3.5. The Ω-conditional base, $\mathscr{C}_\Omega$, is the quotient set of all the Ω-conditional atoms w.r.t. $\simeq$.

Remark 3.6. In the following we will denote the equivalence class of a conditional atom c by c itself. Moreover, any (semantic) subset I of $\mathscr{C}_\Omega$ will implicitly be considered also as an arbitrary (syntactic) Ω-open program $\mu(I)$ obtained by choosing for each equivalence class $c \in I$ an arbitrary element $\mu(c)$ as representative of c. The basic derivation step (unfolding) is independent from the choice of $\mu(I)$, i.e., $\simeq$ is a congruence w.r.t. the unfolding operator. All the semantic operators that we will use on subsets of $\mathscr{C}_\Omega$ (such as T_P^Ω) can be reduced to several applications of this basic step. Therefore, we can define any semantic operator on $I \subseteq \mathscr{C}_\Omega$ in terms of its syntactic counterpart defined in $\mu(I)$, independently from the specific $\mu(I)$. In general, all the

definitions where a syntactic program associated with a subset of $\mathscr{C}_\Omega$ is (implicitly) used will be independent from the choice of the syntactic object. To simplify the notation, we will denote $\mu(I)$ by I and we will denote the syntactic and the semantic operators by the same name.

Let us now give the formal definition of Ω-denotation.

Definition 3.7 (*Ω-denotations*). An Ω-denotation I is any subset of $\mathscr{C}_\Omega$. The set of all the Ω-denotations is denoted by $\mathscr{D}$.

Remark 3.8. In the following the operational semantics $\mathscr{O}_\Omega$ will be formally considered as an Ω-denotation. All the previously stated results for $\mathscr{O}_\Omega(P)$ hold also with such a definition. In fact, in every proof we can consider syntactic clauses as representatives of $\simeq$-equivalence classes, and hence we can obtain a proof for the case of Ω-denotations also. This is correct since, as previously discussed, the derivation step is independent from the choice of the representative of the equivalence class.

Remark 3.9. As shown by the following example, the converse of Corollary 2.11 does not hold, i.e. the semantics $\mathscr{O}_\Omega(P)$ is not fully abstract w.r.t. $\approx_\Omega$. In fact, in general, it is not sufficient to define denotations as sets of equivalence classes of clauses. A full abstractness result (w.r.t. $\approx_\Pi$) was obtained in [31]. However, denotations in [31] are not sets of clauses and the result is obtained by saturating the denotation of a program using essentially the definition of $\approx_\Omega$ (restricted to union with atoms). A similar full abstractness result could be obtained in our case by using the $\approx_\Omega$ equivalence on the domain.

Example 3.10. Let us consider the clauses

c_1 $p(X) :- q(X, b)$,

c_2 $p(X) :- q(X, Y)$,

c_3 $p(X) :- q(X, b), q(Y, a)$

and let us consider the programs $P_1 = \{c_1, c_2\}$ and $P_2 = \{c_2, c_3\}$. Note that for $\Omega = \{q\}$, $\mathscr{O}_\Omega(P_i) = P_i$ for $i = 1, 2$ (considering clauses as $\simeq$-equivalence classes). According to Definition 3.2, $c_1 \not\simeq c_2$, $c_1 \not\simeq c_3$ and $c_2 \not\simeq c_3$, and hence $\mathscr{O}_\Omega(P_1) \neq \mathscr{O}_\Omega(P_2)$. Moreover, $c_1 \not\approx_\Omega c_2$, $c_1 \not\approx_\Omega c_3$ and $c_2 \not\approx_\Omega c_3$. Therefore, no equivalence on single clauses correct w.r.t. $\approx_\Omega$ can be used to identify P_1 and P_2. However, $P_1 \approx_\Omega P_2$ since every answer that can be computed using c_1 can be computed using either c_2 or c_3.

4. Fixpoint semantics

In this section we define a fixpoint semantics $\mathscr{F}_\Omega(P)$ which is proved in Section 4.1, to be equivalent to $\mathscr{O}_\Omega(P)$. This can be achieved by defining an immediate consequence operator T_P^Ω on the lattice $(\mathscr{D}, \subseteq)$ of Ω-denotations. $\mathscr{F}_\Omega(P)$ is the least fixpoint of T_P^Ω.

The immediate consequences operator T_P^Ω is strongly related to the derivation rule used for Ω-open programs. Indeed, we define it in terms of the unfolding rule. As we will show in Section 4.1, this allows an elegant and concise proof of the equivalence between the operational (top-down) and the fixpoint (bottom-up) semantics. Since T_P^Ω models the computed answers in an OR-compositional way, it can be useful for modular (i.e. OR-compositional) bottom-up program analysis.

Definition 4.1 (*Unfolding*). Let P and Q be Ω-open programs. Then the unfolding of P w.r.t. Q is defined as

$$unf_P(Q) = \{(A :- \tilde{L}_1, \ldots, \tilde{L}_n)\vartheta \mid \exists A :- B_1, \ldots, B_n \in P, \exists B_i' :- \tilde{L}_i \in Q, \text{ renamed apart for } i = 1, \ldots, n, \text{ s.t. } \vartheta = mgu((B_1, \ldots, B_n), (B_1', \ldots, B_n'))\}.$$

Example 4.2. Let us consider the following program:

$$\begin{aligned} P = \{ & p(X, Y) :- r(X), s(Y). \\ & r(X) :- s(X). \\ & s(Y) :- q(Y). \\ & s(a). \qquad\qquad \}, \\ unf_P(P) = \{ & p(X, Y) :- s(X), q(Y). \\ & p(X, a) :- s(X). \\ & r(X) :- q(X). \\ & r(a). \\ & s(a). \qquad\qquad \}. \end{aligned}$$

Definition 4.3 (*Immediate consequences operator*) (*Bossi and Menegus* [7]). Let P be an Ω-open program and let I be an Ω-denotation. Then we define

$$T_P^\Omega(I) = unf_P(I \cup Id_\Omega).$$

Note that, for $\Omega = \emptyset$, T_P^Ω is the immediate consequence operator introduced in [18] whose least fixpoint is the s-semantics.

Proposition 4.4 (Bossi and Menegus [7]). *T_P^Ω is monotonic and continuous on the complete lattice $(\mathscr{D}, \subseteq)$.*

The notion of ordinal powers for T_P^Ω is defined as usual, namely $T_P^\Omega \uparrow 0 = \emptyset$, $T_P^\Omega \uparrow n+1 = T_P^\Omega(T_P^\Omega \uparrow n)$ and $T_P^\Omega \uparrow \omega = \bigcup_{n \geqslant 0}(T_P^\Omega \uparrow n)$. Since T_P^Ω is continuous on $(\mathscr{D}, \subseteq)$, well-known results of lattice theory allow one to prove the following proposition.

Proposition 4.5. *$T_P^{\Omega}\uparrow\omega$ is the least fixpoint of T_P^{Ω} on the complete lattice $(\mathscr{D}, \subseteq)$.*

We can then define the fixpoint semantics as follows.

Definition 4.6 (*Bossi and Menegus* [7])(*Fixpoint semantics*). Let P be an Ω-open program. The fixpoint semantics $\mathscr{F}_{\Omega}(P)$ of P is defined as $\mathscr{F}_{\Omega}(P) = T_P^{\Omega}\uparrow\omega$.

Example 4.7. Let P be the program of Example 4.2 and $\Omega = \{q\}$.

$$T_P^{\Omega}\uparrow 1 = unf_P(Id_{\Omega}) = \{s(Y) :- q(Y).\ \ s(a).\},$$

$$T_P^{\Omega}\uparrow 2 = unf_P(T_P^{\Omega}\uparrow 1 \cup Id_{\Omega}) = \{r(X) :- q(X).\ \ r(a).$$
$$s(Y) :- q(Y).\ \ s(a).\},$$

$$T_P^{\Omega}\uparrow 3 = unf_P(T_P^{\Omega}\uparrow 2 \cup Id_{\Omega}) = \{p(X, Y) :- q(X), q(Y).\ \ p(X, a) :- q(X).$$
$$p(a, Y) :- q(Y). \qquad p(a, a).$$
$$r(X) :- q(X). \qquad r(a).$$
$$s(Y) :- q(Y). \qquad s(a). \qquad \}.$$

$$\mathscr{F}_{\Omega}(P) = T_P^{\Omega}\uparrow 3$$

4.1. Unfolding semantics and equivalence results

The equivalence between the operational and the fixpoint semantics can be proved by introducing the intermediate notion of *unfolding semantics* $\mathscr{U}_{\Omega}(P)$ [41, 42, 14]. $\mathscr{U}_{\Omega}(P)$ is obtained as the limit of the (top-down) unfolding process. Since the unfolding rule preserves computed answers in a compositional way, $\mathscr{U}_{\Omega}(P)$ is equivalent to the operational semantics $\mathscr{O}_{\Omega}(P)$. The proof of this equivalence is straightforward, since $\mathscr{O}_{\Omega}(P)$ and $\mathscr{U}_{\Omega}(P)$ are based on the same inference rule (applied in sequence and in parallel, respectively). On the other hand, the equivalence between $\mathscr{U}_{\Omega}(P)$ and the bottom-up semantics $\mathscr{F}_{\Omega}(P)$ can be based on the relation $T_P^{\Omega}(I) = unf_P(I \cup Id_{\Omega})$. Let us first formally define the unfolding semantics.

Definition 4.8. Let P be a set of clauses. Then we define

$$\iota_{\Omega}(P) = \{c \in P \mid c \in \mathscr{C}_{\Omega}\}.$$

Definition 4.9 (*Unfolding semantics*). Let P be an Ω-open program. Then we define the collection of programs

$$P_1 = P,$$

$$P_{n+1} = unf_{P_n}(P \cup Id_{\Omega}).$$

The unfolding semantics $\mathscr{U}_\Omega(P)$ of the program P is defined as

$$\mathscr{U}_\Omega(P)=\bigcup_{n=1,2,\cdots} \iota_\Omega(P_n).$$

Example 4.10. Let P be the program of Example 4.2 and $\Omega=\{q\}$.

$$P_1=P,$$

$$P_2=unf_{P_1}(P\cup Id_\Omega)=\{p(X,Y):-s(X),q(Y).\quad p(X,a):-s(X).$$
$$r(X):-q(X).\qquad r(a).$$
$$s(Y):-q(Y).\qquad s(a).\qquad\},$$

$$P_3=unf_{P_2}(P\cup Id_\Omega)=\{p(X,Y):-q(X),q(Y).\quad p(a,Y):-q(Y).$$
$$p(X,a):-q(X).\qquad p(a,a).$$
$$r(X):-q(X).\qquad r(a).$$
$$s(Y):-q(Y).\qquad s(a).\qquad\},$$

$$\mathscr{U}_\Omega(P)=P_3.$$

In order to prove the equivalence we need two lemmata. The first one states the associativity of the unfolding operator.

Lemma 4.11 (Denis and Dalahaye [14]). *Let P,Q,W be programs. Then $unf_P(unf_Q(W))=unf_{unf_P(Q)}(W)$.*

Lemma 4.12. *Let P be an Ω-open program and let P_n be as in Definition 4.9. Let W be a set of clauses and let us define $unf_P^1(W)=unf_P(W)$ and, for $n>1$, $unf_P^n(W)=unf_P(unf_P^{n-1}(W))$. Then, for $n\geqslant 1$, we have*

(1) $unf_P(unf_P^{n-1}(W))=unf_P^{n-1}(unf_P(W))$,
(2) $unf_{P_{n+1}}(W)=unf_P(unf_{P\cup Id_\Omega}^n(W))$,
(3) $unf_{P\cup Id_\Omega}^n(Id_\Omega)=Id_\Omega\cup T_P^\Omega\uparrow n$,
(4) $T_P^\Omega\uparrow n=\iota_\Omega(P_n)$.

Proof. All the proofs are by induction on n.

(1) Straightforward.

(2) For $n=1$, we have the equalities

$$unf_{P_2}(W)=unf_{unf_P(P\cup Id_\Omega)}(W)\quad\text{(by Definition 4.9)}$$
$$=unf_P(unf_{P\cup Id_\Omega}(W))\quad\text{(by Lemma 4.11).}$$

For $n \geqslant 1$, assume $unf_{P_n}(W) = unf_P(unf^{n-1}_{P \cup Id_\Omega}(W))$. Then

$$unf_{P_{n+1}}(W) = unf_{unf_{P_n}(P \cup Id_\Omega)}(W) \quad \text{(by Definition 4.9)}$$

$$= unf_{P_n}(unf_{P \cup Id_\Omega}(W)) \quad \text{(by Lemma 4.11)}$$

$$= unf_P(unf^{n-1}_{P \cup Id_\Omega}(unf_{P \cup Id_\Omega}(W))) \quad \text{(by inductive hypothesis)}$$

$$= unf_P(unf^{n}_{P \cup Id_\Omega}(W)) \quad \text{(by definition of } unf^n \text{ and part 1).}$$

and the thesis holds.

(3) For $n = 1$, we have the equalities

$$unf_{P \cup Id_\Omega}(Id_\Omega) = unf_P(Id_\Omega) \cup unf_{Id_\Omega}(Id_\Omega) \quad \text{(by definition of unfolding)}$$

$$= unf_P(Id_\Omega) \cup Id_\Omega \quad \text{(by definition of } Id_\Omega)$$

$$= Id_\Omega \cup T_P^\Omega \uparrow 1 \quad \text{(by definition of } \uparrow \text{ and by Definition 4.3).}$$

For $n \geqslant 1$, assume $unf^n_{P \cup Id_\Omega}(Id_\Omega) = Id_\Omega \cup T_P^\Omega \uparrow n$.

$$unf^{n+1}_{P \cup Id_\Omega}(Id_\Omega) = unf_{P \cup Id_\Omega}(unf^n_{P \cup Id_\Omega}(Id_\Omega)) \quad \text{(by definition of } unf^{n+1})$$

$$= unf_{P \cup Id_\Omega}(Id_\Omega \cup T_P^\Omega \uparrow n) \quad \text{(by inductive hypothesis)}$$

$$= unf_P(Id_\Omega \cup T_P^\Omega \uparrow n) \cup unf_{Id_\Omega}(Id_\Omega \cup T_P^\Omega \uparrow n)$$
(by definition of unfolding)

$$= unf_P(Id_\Omega \cup T_P^\Omega \uparrow n) \cup unf_{Id_\Omega}(T_P^\Omega \uparrow n) \cup Id_\Omega$$
(by definition of Id_Ω)

$$= T_P^\Omega \uparrow n+1 \cup Id_\Omega \cup unf_{Id_\Omega}(T_P^\Omega \uparrow n)$$
(by definition of $\uparrow$ and by Definition 4.3)

$$= T_P^\Omega \uparrow n+1 \cup Id_\Omega \quad \text{(by the following remark).}$$

The last equality holds because, by definition of Id_Ω and the unfolding rule, $unf_{Id_\Omega}(T_P^\Omega \uparrow n) \subseteq T_P^\Omega \uparrow n$ and, since T_P^Ω is monotonic, $T_P^\Omega \uparrow n \subseteq T_P^\Omega \uparrow n+1$.

(4) First note that, by Definitions 4.9 and 4.3, $\iota_\Omega(P_n) = unf_{P_n}(Id_\Omega) = T^\Omega_{P_n}(\emptyset)$. Since $P_1 = P$, $\iota_\Omega(P_1) = T_P^\Omega(\emptyset) = T_P^\Omega \uparrow 1$. For $n > 1$ note that, by definition of $\uparrow$ and by Definition 4.3, $T_P^\Omega \uparrow n = unf_P(Id_\Omega \cup T_P^\Omega \uparrow n-1)$. Then for $n > 1$ the following equalities hold:

$$T_P^\Omega \uparrow n = unf_P(Id_\Omega \cup T_P^\Omega \uparrow n-1) \quad \text{(by the previous remark)}$$

$$= unf_P(unf^{n-1}_{P \cup Id_\Omega}(Id_\Omega)) \quad \text{(by part 2 of Lemma 4.12)}$$

$$= unf_{P_n}(Id_\Omega) \quad \text{(by part 1 of Lemma 4.12)}$$

$$= \iota_\Omega(P_n) \quad \text{(by the previous remark)}$$

and this completes the proof. □

The following theorem shows the equality of the unfolding and the fixpoint semantics.

Theorem 4.13. *Let P be a program. Then $\mathscr{F}_\Omega(P) = \mathscr{U}_\Omega(P)$.*

Proof. By definition, $\mathscr{F}_\Omega(P) = T_P^\Omega \uparrow \omega$. Note that $c \in T_P^\Omega \uparrow \omega$ iff $\exists n$ such that $c \in T_P^\Omega \uparrow n$ and $c \in \mathscr{U}_\Omega(P)$ iff $\exists n$ such that $c \in \iota_\Omega(P_n)$. Then the thesis follows from part (4) of Lemma 4.12. □

The following theorem states the equality of the unfolding and the operational semantics. The proof (in the appendix) is straightforward since both the semantics are top-down and are based on the same inference rule.

Theorem 4.14. *Let P be an Ω-open program. Then $\mathscr{O}_\Omega(P) = \mathscr{U}_\Omega(P)$.*

Corollary 4.15 (Equivalence). *Let P be a program. Then $\mathscr{F}_\Omega(P) = \mathscr{O}_\Omega(P)$.*

Proof. Straightforward by Theorems 4.13 and 4.14. □

As previously discussed, the semantics $\mathscr{F}_\Omega(P)$ is $\approx_\Omega$-equivalent to the program P, i.e. given a program P_1 such that $P \cup_\Omega P_1$ is defined, each answer computed in $P \cup_\Omega P_1$ can be computed also in $\mathscr{F}_\Omega(P) \cup_\Omega P_1$. Note that such a semantics is also closed under unfolding, i.e. it contains all the results of (partial) unfoldings performed by using clauses in it (see Definition 4.16). As a consequence, given a set of atoms Q with $Pred(Q) \subseteq \Omega$, each answer computed in $P \cup_\Omega Q$ can be computed in $\mathscr{F}_\Omega(P) \cup_\Omega Q$ by using a single "parallel" derivation step (note that, by Theorem 5.20, considering only atoms is not a restriction). The following propositions formalize these properties. Recall that by Π we denote the set of all the predicate symbols.

Definition 4.16. A set of clauses (a subset of $\mathscr{C}_\Omega$) I is *u*-closed iff $unf_I(I \cup Id_\Pi) \subseteq I$.

Lemma 4.17. *Let P be a program. Then $T_P^\Omega \uparrow \omega$ is u-closed.*

Proof. To simplify the notation let us denote $T_P^\Omega \uparrow \omega$ by F_ω and $T_P^\Omega \uparrow n$ by F_n. By Definition 4.16, we have to prove that $unf_{F_\omega}(F_\omega \cup Id_\Pi) = F_\omega$. We show the two inclusions separately.

$unf_{F_\omega}(F_\omega \cup Id_\Pi) \subseteq F_\omega$: Note that $c \in F_\omega$ iff $\exists n$ such that $c \in F_n$. We then prove, by induction on n, that $\forall n \geqslant 0$, $unf_{F_n}(F_\omega \cup Id_\Pi) \subseteq F_\omega$.

For $n = 0$, the proof is straightforward, since $F_0 = \emptyset$ and $unf_\emptyset(F_\omega \cup Id_\Pi) = \emptyset$.

For $n>0$, we have

$$unf_{F_n}(F_\omega \cup Id_\Pi) = unf_P(unf_{F_{n-1}\cup Id_\Omega}(F_\omega \cup Id_\Pi))$$
(by definition of $\uparrow$ and by Lemma 4.11)

$$\subseteq unf_P(F_\omega \cup Id_\Omega)$$
(by inductive hypothesis and by definition of Id_Ω)

$$= F_\omega \quad \text{(by Definition 4.3 and since } F_\omega \text{ is a fixpoint).}$$

$unf_{F_\omega}(F_\omega \cup Id_\Pi) \supseteq F_\omega$: We prove, by induction on n, that $\forall n \geqslant 0$, $unf_{F_\omega}(F_\omega \cup Id_\Pi) \supseteq F_n$.
For $n=0$, the proof is straightforward, since $F_0 = \emptyset$.
For $n>0$, we have

$F_n = unf_P(F_{n-1} \cup Id_\Omega)$ (by Definition 4.3)

$\subseteq unf_P(unf_{F_\omega}(F_\omega \cup Id_\Pi) \cup Id_\Omega)$ (by inductive hypothesis)

$\subseteq unf_P(unf_{F_\omega \cup Id_\Omega}(F_\omega \cup Id_\Pi))$ (by definition of unfolding)

$= unf_{unf_P(F_\omega \cup Id_\Omega)}(F_\omega \cup Id_\Pi)$ (by Lemma 4.11)

$= unf_{F_\omega}(F_\omega \cup Id_\Pi)$ (by Definition 4.3 and since F_ω is a fixpoint)

and this concludes the proof. □

Proposition 4.18. *Let $I \subseteq \mathscr{C}_\Omega$. Then $T_I^\Omega \uparrow \omega$ is the least (w.r.t. set inclusion) subset of $\mathscr{C}_\Omega$ which is u-closed and which contains I.*

Proof. $T_I^\Omega \uparrow \omega$ is u-closed by Lemma 4.17. Note that, by Definition 4.3, for $I \subseteq \mathscr{C}_\Omega$, $I = T_I^\Omega(\emptyset)$. Then by definition of $\uparrow$, $I \subseteq T_I^\Omega \uparrow \omega$. Let $H \subseteq \mathscr{C}_\Omega$, such that H is u-closed and $I \subseteq H$. Let us show by induction on n that $\forall n$, $T_I^\Omega \uparrow n \subseteq H$.
For $n=0$, it is obvious since $T_I^\Omega \uparrow 0 = \emptyset$.
For $n>0$, let us assume by inductive hypothesis that $T_I^\Omega \uparrow n \subseteq H$. Then

$T_I^\Omega \uparrow n+1 = T_I^\Omega(T_I^\Omega \uparrow n)$ (by definition of $\uparrow$)

$\subseteq T_I^\Omega(H)$ (by inductive hypothesis)

$\subseteq T_H^\Omega(H)$ (since $I \subseteq H$)

$= H$ (since H is u-closed).

Since $T_I^\Omega \uparrow \omega = \bigcup_{n<\omega} T_I^\Omega \uparrow n$, the thesis holds. □

Lemma 4.19. *Let P be a u-closed set of clauses and let Q be a set of atoms with $Pred(Q) \subseteq \Omega$. Then, for any $n>1$, $T_{P\cup Q}^\emptyset \uparrow n = T_P^\emptyset(Q) \cup Q$.*

Proof. The proof is by induction on n. For $n=2$, by Definition 4.3, we have to prove that

$$unf_{P\cup Q}(unf_{P\cup Q}(\emptyset))=unf_P(Q)\cup Q.$$

We show the two inclusions separately. By definition of unfolding,

$$unf_{P\cup Q}(unf_{P\cup Q}(\emptyset))=unf_P(unf_P(\emptyset))\cup Q)\cup Q\supseteq unf_P(Q)\cup Q.$$

On the other hand,

$$\begin{aligned}
unf_{P\cup Q}(unf_{P\cup Q}(\emptyset)) &= unf_P(unf_P(\emptyset)\cup Q)\cup Q && \text{(by definition of unfolding)}\\
&\subseteq unf_P(unf_P(Q)\cup Q)\cup Q && \text{(by definition of unfolding)}\\
&= unf_P(unf_{P\cup Id_\Pi}(Q))\cup Q && \text{(by definition of } Id_\Pi)\\
&= unf_{unf_P(P\cup Id_\Pi)}(Q)\cup Q && \text{(by Lemma 4.11)}\\
&= unf_P(Q)\cup Q && \text{(since } P \text{ is } u\text{-closed)}
\end{aligned}$$

and the thesis holds for the base case. The proof for the inductive case is identical and thus omitted. □

Proposition 4.20. *Let P be an Ω-open program and let Q be a set of atoms with $Pred(Q)\subseteq\Omega$. Then for any goal $G=A_1,\ldots,A_n$, $G\overset{\vartheta}{\rightsquigarrow}_{P\cup_u Q}\square$ iff there exists $B_1,\ldots,B_n\in T^\emptyset_{\mathscr{F}_\Omega(P)}(Q)\cup Q$ such that $\gamma=mgu((A_1,\ldots,A_n),(B_1,\ldots,B_n))$ and $\gamma|_{Var(G)}=\vartheta|_{Var(G)}$.*

Proof. The completeness theorem of s-semantics [18] can be obtained as the particular case $\Omega=\emptyset$ of Theorem 2.10 and can be stated as follows. $G\overset{\vartheta}{\rightsquigarrow}_P\square$ iff there exists $B_1,\ldots,B_n\in T^\emptyset_P\uparrow\omega$ such that $\gamma=mgu((A_1,\ldots,A_n)(B_1,\ldots,B_n))$ and $\gamma|_{Var(G)}=\vartheta|_{Var(G)}$. By definition of $\mathscr{F}_\Omega(P)$ (and a straightforward inductive argument), $T^\emptyset_{P\cup Q}\uparrow\omega=T^\emptyset_{\mathscr{F}_\Omega(P)\cup Q}\uparrow\omega$. Since $\mathscr{F}_\Omega(P)$ is u-closed, by Lemma 4.19 and by definition of $\uparrow$, $T^\emptyset_{\mathscr{F}_\Omega(P)\cup Q}\uparrow\omega=T^\emptyset_{\mathscr{F}_\Omega(P)}(Q)\cup Q$ and the thesis holds. □

5. Model theory

As we have shown, the operational and fixpoint semantics of a program P define an Ω-denotation I_P. Let J be any set of ground atoms defining the open predicates in P. Then (the Herbrand model) J can be viewed as a program which closes P, i.e. which completely specifies the open predicates of P. Now I_P is a representative, in terms of clauses, of a function which when applied to a Herbrand model J closing P, returns the least Herbrand model of the composition $P\cup J$. If we consider a class of possible closures J of P, we can view I_P as a syntactic notation for the set $\mathscr{H}(I_P)$ of the least Herbrand models of all the programs $P\cup J$. By abuse of notation, we will then say that

I_P is a "model" (Ω-model) of P iff all the Herbrand interpretations in $\mathscr{H}(I_P)$ are models of P. Depending on which sets J we allow to use as closures for Ω-denotations we obtain different classes of Ω-models.

In this section, we first define Ω-models which are obtained by using a restricted definition $\mathscr{H}'(I_P)$. Such a restriction allows to obtain a large class of Ω-models which includes the standard Herbrand models. We show that the least upper bound of the u-closed Ω-models of a program P (according to a suitable ordering) is the least Herbrand model of P. We then introduce a particular class of Ω-models which are obtained by using the more general definition of $\mathscr{H}(I_P)$, and hence are compositional w.r.t. $\cup_\Omega$. We show that $\mathscr{F}_\Omega(P)$ is a compositional Ω-model. Finally, we prove the full abstractness of a (compositional) model-theoretic semantics w.r.t. the equivalence induced by the successful derivation notion of observable.

5.1. *Ω-models*

In the following we denote by $\mathscr{B}_\Omega$ the Ω-denotation $\{p(\tilde{t}) \mid p \in \Omega\}$. We denote by $M(K)$ the least Herbrand model of the set of clauses K. Moreover, if I is an Ω-denotation, we denote also by $M(I)$ the least Herbrand model of $I' = \{\mu(c) \mid c \in I\}$, where $\mu(c)$ is any element of the equivalence class c. Clearly, $M(I)$ is well defined since it does not depend on the element chosen in the $\simeq$ equivalence classes. Analogously, we call Herbrand model of I any Herbrand model of I'.

Let us now define the Ω-models.

Definition 5.1. Let I be an Ω-denotation for an Ω-open program. Then

$$\mathscr{H}'(I) = \{M(I \cup J) \mid J \subseteq A_\Omega(I)\},$$

where $A_\Omega(I) = \{p(\tilde{t}) \mid p \in \Omega$ and $p(\tilde{t})$ is a ground instance of an atom, in a body of a clause in $I\}$.

Example 5.2. Let $I = \{p(a) :- q(b)\}$ be an Ω-denotation. Then

(1) for $\Omega = \{q\}$ $\quad \mathscr{H}'(I) = \{\emptyset, \{p(a), q(b)\}\}$,

(2) for $\Omega = \{p, q\}$ $\quad \mathscr{H}'(I) = \{\emptyset, \{p(a), q(b)\}\}$.

Definition 5.3 (*Ω-model*). Let P be an Ω-open program and I be an Ω-denotation. I is an Ω-model of P iff $\forall J \in \mathscr{H}'(I)$, J is a Herbrand model of P.

Let us show an example of Ω-model.

Example 5.4. Let us consider the following program:

$$\begin{aligned} P = \{ & q(X) :- p(X) \\ & r(X) :- s(X). \\ & s(b). \\ & p(a). \qquad \}. \end{aligned}$$

Let us assume $\Omega=\{p\}$ and consider the Ω-denotation $\mathscr{O}_\Omega(P)$, which, according to Definition 2.5 is

$$\mathscr{O}_\Omega(P)=\{q(X):-p(X), p(a), q(a), r(b), s(b)\}.$$

By Definition 5.1, $\mathscr{H}'(\mathscr{O}_\Omega(P_1))=\{H_1, H_2, H_3, \ldots\}$, where, denoting by $[p(X)]$ the set of ground instances of $p(X)$,

$$H_1=\{p(a), q(a), r(b), s(b)\},$$

$$H_2=\{p(a), p(b), q(a), q(b), r(b), s(b)\},$$

$$\vdots$$

$$H_\omega=\{r(b), s(b)\}\cup[p(X)]\cup[q(X)].$$

Since $H_1, H_2, \ldots, H_\omega$ are all Herbrand models of P, by Definition 5.3 $\mathscr{O}_\Omega(P)$ is an Ω-model of P_1.

In general, we can prove that $\mathscr{O}_\Omega(P)$ is an Ω-model of P. The proof follows from a more general result that we will give in Section 5.2 (see corollary 5.24).

Note that in Definition 5.1 we impose a restriction on the set J of ground atoms which are added to the Ω-denotation I in order to completely specify the "open" predicates. Namely, we require that J contains only atoms which unify with atoms already in bodies of clauses in I. The reason for such a restriction is that we want that standard Herbrand models are also Ω-models (Proposition 5.5). Clearly, if M is a Herbrand model of the program P and N is an arbitrary set of ground atoms, $M\cup N$ could be not a model of P. Therefore, as shown by Example 5.6, if we drop our restriction in Definition 5.1, Proposition 5.5 does not hold anymore.

Proposition 5.5. *Let P be an Ω-open program. Then every Herbrand model of P is an Ω-model of P.*

Proof. The proof is straightforward, since for any Herbrand interpretation I, $\mathscr{H}'(I)=\{I\}$. □

Example 5.6. Let us consider the Ω-open program $P=\{p(a):-q(a)\}$, where $\Omega=\{q\}$. Then $\emptyset$ is a (the least) Herbrand model of P. If, by violating the condition $J\subseteq A_\Omega(I)$, $\{q(a)\}\in\mathscr{H}'(\emptyset)$, $\emptyset$ would not be an Ω-model of P because $\{q(a)\}$ is not a Herbrand model of P.

A relevant property of standard Herbrand models states that the intersection of a set of models of a program P is always a model of P. This allows one to define the model-theoretic semantics of P as the least Herbrand model obtained by intersecting all the Herbrand models of P. The following example shows that this important

property does not hold any more when considering Ω-models with set-theoretic operations.

Example 5.7. Let $\Omega=\{q\}$ and P be the following Ω-open program:

$$P=\{p(b):-q(b)$$
$$p(X)$$
$$q(a) \qquad \}.$$

The $\mathcal{O}_\Omega(P)=\{p(b):-q(b), p(X), q(a)\}$ and $M(P)=\{q(a)\}\cup\{p(\tilde{t})\mid \tilde{t} \text{ is a ground term}\}$. By Definition 5.3 $\mathcal{O}_\Omega(P)$ and $M(P)$ are Ω-models of P. However, $\mathcal{O}_\Omega(P)\cap M(P)=\{q(a)\}$ is not an Ω-model of P.

Intuitively, the Ω-model intersection property does not hold because set-theoretic operations do not adequately model the operations on conditional atoms. Namely, the information of an Ω-denotation I_1 may be contained in I_2 without I_1 being a subset of I_2. In order to define the model-theoretic semantics for Ω-open programs as a unique least u-closed Ω-model, we then need a suitable partial order $\sqsubseteq$ on Ω-denotations. $\sqsubseteq$ should model the meaning of Ω-denotations, in such a way that $(\mathcal{D}, \sqsubseteq)$ is a complete lattice. This can be obtained by considering $\sqsubseteq$ as given in Definition 5.8. It is worth noting that the least u-closed Ω-model is the standard least Herbrand model (Proposition 5.17). Moreover, the most expressive Ω-model $\mathcal{O}_\Omega(P)$ is a nonminimal Ω-model (see Section 5.2). The following definitions are similar to those given in [20] for the noncompositional semantics of positive logic programs.

Definition 5.8. Let I_1, I_2 be Ω-denotations. We define:
- $I_1 \leqslant_d I_2$ iff $\forall c_1\in I_1\ \exists c_2\in I_2$ such that $c_2 \leqslant_c c_1$.
- $I_1\sqsubseteq I_2$ iff $(I_1 \leqslant_d I_2)$ and $(I_2 \leqslant_d I_1$ implies $I_1\subseteq I_2)$.

where $\leqslant_c$ is defined in Definition 3.2.

The proof of the following proposition is the same of the analogous in [20].

Proposition 5.9. *The relation $\leqslant_d$ is a preorder and the relation $\sqsubseteq$ is an ordering.*

Note that if $I_1\subseteq I_2$, then $I_1\sqsubseteq I_2$, since $I_1\subseteq I_2$ implies $I_1 \leqslant_d I_2$. To show that the set of Ω-denotations with $\sqsubseteq$ is a complete lattice we need two more propositions.

Definition 5.10. Let I be an Ω-denotation. We define

$$Min(I)=\{c\in I\mid \forall c'\in I, c' \leqslant_c c \Rightarrow c'=c\}$$

Example 5.11. (a) If $I=\{p(X), p(a):-q(b), q(b), p(a)\}$ then $Min(I)=\{p(X), q(b)\}$,
(b) $J=\{q(X):-p(X), r(X)$

$q(b):-p(b)$

$q(b):-p(X)$

$r(b)$ $\}$,

$Min(J)=\{q(X):-p(X), r(X)$

$q(b):-p(X)$

$r(b)$ $\}$.

Proposition 5.12. Let I_1 and I_2 be Ω-denotations. Then $I_1 \leqslant_d I_2$ and $I_2 \leqslant_d I_1$ iff $Min(I_1)=Min(I_2)$.

Proof. "If": First observe that, by Definition 3.2, if $c_1=H_1:-B_1,\ldots,B_n$, $c_2=H_2:-D_1,\ldots,D_m$ and $c_1 \leqslant_c c_2$, then there exists a substitution σ such that $H_1\sigma=H_2$ and the multiset $\{^+B_1\sigma,\ldots,B_n\sigma\}^+$ is contained in the multiset $\{^+D_1,\ldots,D_m\}^+$. Considering bodies as multisets, given a clause c there exists only a finite number up to renaming of clauses c' which are less instantiated than c. Moreover, the number of the atoms in the body of a clause is finite. Therefore, there exists only a finite number of clauses which are $\leqslant_c$ than the clause c and which are not equal up to renaming. Since an Ω-denotation I contains $\simeq$ equivalences classes of clauses, I contains no infinite descending chains $c_1 >_c c_2 >_c c_3 \ldots$ (where $c_i >_c c_j$ iff $c_j \leqslant_c c_i$ and $c_j \neq c_i$). Then the thesis follows from Definitions 5.10 and 5.8.

Only if: Assume $Min(I_1) \neq Min(I_2)$ and let $c_1 \in Min(I_1) \setminus Min(I_2)$. Since $I_1 \leqslant_d I_2$, $\exists c_2 \in Min(I_2)$ such that $c_2 \neq c_1$ and $c_2 \leqslant_c c_1$. Moreover, since $I_2 \leqslant_d I_1$, $\exists c \in Min(I_1)$ such that $c \leqslant_c c_2 \neq c_1$. Then $c \neq c_1$ (recall that the c_i's are equivalence classes), which contradicts the hypothesis $c_1 \in Min(I_1)$. □

Definition 5.13. Let Λ be a set of Ω-denotations. We introduce the following notations.

- $\nabla\Lambda=\bigcup_{I\in\Lambda} I$,
- $Min(\Lambda)=Min(\nabla\Lambda)$,
- $\bigsqcup\Lambda=A$ *where* $A=Min(\Lambda)\cup\nabla\{I\in\Lambda \mid Min(\Lambda)\subseteq I\}$.

Note that $Min(\Lambda)=Min(\bigsqcup\Lambda)$. We can now prove the following proposition. The proof is essentially that one in [20], rephrased by using Proposition 5.12.

Proposition 5.14. *For any set Λ of Ω-denotations there exists the least upper bound of Λ, $lub(\Lambda)$, and $lub(\Lambda)=\bigsqcup\Lambda$ holds.*

Proof. (1) $\bigsqcup\Lambda$ *is an upper bound of* Λ: If I is an element of Λ, then $I \leqslant_d \bigsqcup\Lambda$; in fact $\forall c \in I$, $c \in \bigcup_{I\in\Lambda} I$, thus $\exists c' \in Min(\Lambda)$ such that $c' \leqslant_c c$, and so $I \leqslant_d \bigsqcup\Lambda$, since $Min(\Lambda) \subseteq \bigsqcup\Lambda$. Moreover, if $\bigsqcup\Lambda \leqslant_d I$, then by Proposition 5.12, $Min(\Lambda) = Min(\bigsqcup\Lambda) = Min(I) \subseteq I$. Therefore, by definition of $\bigsqcup\Lambda$, $I \subseteq \bigsqcup\Lambda$.

(2) $\bigsqcup\Lambda$ *is the least upper bound of* Λ: Let H be an upper bound of Λ. Since, for every $I \in \Lambda$, $I \leqslant_d H$, then $\bigcup_{I\in\Lambda} I \leqslant_d H$ and $\bigsqcup\Lambda \leqslant_d H$. Assume now $H \leqslant_d \bigsqcup\Lambda$, then $Min(H) = Min(\bigsqcup\Lambda)$, and therefore $Min(\Lambda) = Min(\bigsqcup\Lambda) \subseteq H$. Moreover, for any $I \in \Lambda$ such that $Min(\Lambda) \subseteq I$, $Min(H) \subseteq I$, and therefore (since $I \sqsubseteq H$) $I \subseteq H$. Then $\bigvee\{I \in \Lambda \mid Min(\Lambda) \subseteq I\} \subseteq H$, and therefore $\bigsqcup\Lambda \subseteq H$ holds. □

Proposition 5.15. *The poset of all the* Ω*-denotations* $(\mathscr{D}, \sqsubseteq)$ *is a complete lattice with lub operator given by* $\bigsqcup X$ *(Definition* 5.13). $\mathscr{C}_\Omega$ *is the top element and* $\emptyset$ *is the bottom element.*

Proof. For any set Λ of Ω-denotations, the existence of its least upper bound is ensured by Proposition 5.14. The greatest lower bound of Λ is then given by $glb(\Lambda) = lub(\{I \in \mathscr{D} \mid \forall I' \in \Lambda,\ I \sqsubseteq I'\})$. □

We show now that according to $\sqsubseteq$ ordering, the least u-closed Ω-model is the standard least Herbrand model. This fact justifies our choice of the ordering relation. Note that, also with the $\sqsubseteq$ ordering the *glb* of a set of Ω-models is not an Ω-model. However, for the set of all the u-closed Ω-models such a property holds. Recall that an u-closed set (Definition 4.16) of clauses S contains the set S_u of all the unit clauses which can be obtained by iterated unfoldings of clauses in S. The least Herbrand model of S can then be obtained by considering the ground instances of the unit clauses in S. As a consequence, each u-closed set is greater, according to $\sqsubseteq$ ordering, than its least Herbrand model. This allows us to obtain the result in Proposition 5.17.

Proposition 5.16. *For any u-closed* Ω*-model* I *of a program* P *there exists a standard Herbrand model* I' *of* P *such that* $I' \sqsubseteq I$.

Proof. Define $I' = M(I) \in \mathscr{H}'(I)$. Then, I' is a standard Herbrand model of P. We show now that $I' \sqsubseteq I$. By definition of $\mathscr{H}'(I)$, since I is u-closed, $I' \leqslant_d I$. Assume now $I \leqslant_d I'$. Then $I' = Min(I') = Min(I) \subseteq I$ (by Proposition 5.12). □

Proposition 5.17. *Let* P *be a program. Then* $glb(\{I \in \mathscr{D} \mid I$ *is an* Ω*-model of* $P\}) = M(P)$ *(the least standard Herbrand model of* P*).*

Proof. Note that the standard Herbrand models are ordered by set inclusion, then apply Propositions 5.9 and 5.16. □

5.2. Compositional models

As previously discussed, when considering model theory we are no more concerned with computed answers, whose meaning cannot be reflected by purely logical notions such as truth, logical consequence, etc. However, we can still be interested in the "compositional" aspects of the (model-theoretic) semantics. Note that Ω-models are not compositional w.r.t. $\cup_\Omega$. Indeed, the standard least Herbrand model $M(P)$ is an Ω-model (Proposition 5.5) and, as shown by Example 2.1, in general, $M(P_1 \cup P_2)$ cannot be obtained from $M(P_1)$ and $M(P_2)$. Moreover, as shown in Example 5.6, the composition of an Ω-model of P_1 and of an Ω-model of P_2 is not an Ω-model of $P_1 \cup P_2$.

Among Ω-models we could then be interested in identifying a particular class of "compositional Ω-models" for which the above properties hold. This can be achieved by using the more general and natural definition of $\mathscr{H}(I)$.

Definition 5.18 (*compositional Ω-model*). Let P be an Ω-open program and I be an Ω-denotation. We define

$$\mathscr{H}(I) = \{M(I \cup J) \mid J \subseteq \mathscr{B}_\Omega\}.$$

Then I is a *compositional Ω-model* of P iff $\forall J \in \mathscr{H}(I)$, J is a Herbrand model of P.

Clearly, since $\mathscr{H}'(I) \subseteq \mathscr{H}(I)$, if I is a *compositional Ω-model* then it is also an Ω-model according to Definition 5.3. The vice versa does not hold. For example, in the program $P = \{p(a) :- q(a)\}$ with $\Omega = \{q\}$ of Example 5.6, $\emptyset$ (the least Herbrand model of P) is an Ω-model but is not a compositional Ω-model since $\{q(a)\} \in \mathscr{H}(\emptyset)$.

In the following we will show some properties of compositional Ω-models. We first need the formal definition of the equivalence obtained by considering successful derivations as observable and $\cup_\Omega$ as composition.

Definition 5.19. Let P_1, P_2 be Ω-open programs. Then $P_1 \approx_{\Omega,g} P_2$ iff for any Ω-open program Q such that $P_1 \cup_\Omega Q$ and $P_2 \cup_\Omega Q$ are defined and for any goal G, $G \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup_\Omega Q} \square$ iff $G \overset{\gamma}{\rightsquigarrow}_{P_2 \cup_\Omega Q}$. □

The following theorem generalizes the result given in [31] for the case $\Omega = \Pi$.

Theorem 5.20 (Gabbrielli et al. [25]). *Let P_1, P_2 be programs. Then $P_1 \not\approx_\Omega P_2$ iff there exists a set of atoms $Q \subseteq \mathscr{B}_\Omega$ such that $P_1 \cup_\Omega Q \not\approx_\emptyset P_2 \cup_\Omega Q$. Moreover, $P_1 \not\approx_{\Omega,g} P_2$ iff there exists a set of (ground) atoms $Q \subseteq \mathscr{B}_\Omega$ such that $P_1 \cup_\Omega Q \not\approx_{\emptyset,g} P_2 \cup_\Omega Q$.*

Corollary 5.21 (Gabbrielli et al. [25]). *Let P_1 and P_2 be Ω-open programs and let assume that programs are defined on a signature containing infinite constant symbols. Then for any $Q \subseteq \mathscr{B}_\Omega$ and for any goal G, $G \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup_\Omega Q} \square$ implies $G \overset{\gamma}{\rightsquigarrow}_{P_2 \cup_\Omega Q} \square$ iff for every set $Q \subseteq \mathscr{B}_\Omega$ of (ground) atoms $M(P_1 \cup Q) \subseteq M(P_2 \cup Q)$.*

The following proposition shows that compositional Ω-models satisfy the previously sketched compositional properties. Note that the proposition does not hold for Ω-models.

Proposition 5.22. *Let P_1 be an Ω_1-open program and P_2 be an Ω_2-open program such that $P_1 \cup_\Omega P_2$ is defined. Let M_i be a compositional Ω_i-model of P_i such that $Pred(M_i) \subseteq Pred(P_i)$, $i=1,2$. Then*

(1) *$M_1 \cup_\Omega M_2$ is defined and is a compositional Ω-model of $P_1 \cup_\Omega P_2$.*

(2) *Let M'_1 be a compositional Ω_1-model of P_1. Then for any $Q \subseteq \mathscr{B}_{\Omega_1}$ and for any goal G, $G \overset{\vartheta}{\rightsquigarrow}_{P_1 \cup_\Omega Q} \square$ implies $G \overset{\gamma}{\rightsquigarrow}_{M'_1 \cup_\Omega Q} \square$.*

Proof. (1) Since $P_1 \cup_\Omega P_2$ is defined, then $\Omega_1 \cup \Omega_2 \supseteq \Omega$ and $Pred(M_1) \cap Pred(M_2) \subseteq Pred(P_1) \cap Pred(P_2) \subseteq \Omega_1 \cap \Omega_2$. Then $M_1 \cup_\Omega M_2$ is defined. Now, let $J \in \mathscr{H}(M_1 \cup_\Omega M_2)$. By definition of $\mathscr{H}$, $J = M(M_1 \cup M_2 \cup I)$, where $I \subseteq \mathscr{B}_\Omega$. Observe that J is a Herbrand model of P_1 iff $J \cap \mathscr{B}_{Pred(P_1)} = M(M_1 \cup (J \cap \mathscr{B}_{Pred(P_1)}))$ $= M(M_1 \cup I_1)$ is a Herbrand model of P_1, where $I_1 = J \cap \mathscr{B}_{\Omega_1}$ (the last equality holds since $\forall A :- \tilde{L} \in M_1$, $Pred(\tilde{L}) \subseteq \Omega_1$). Then $M(M_1 \cup I_1) \in \mathscr{H}(M_1)$ and since M_1 is a compositional Ω_1-model of P_1, then $M(M_1 \cup I_1)$ and, by the previous observation, J are Herbrand models of P_1. Analogously we can prove that J is a Herbrand model of P_2. Therefore, J is a Herbrand model of $P_1 \cup P_2$ and this completes the proof.

(2) Straightforward, by Corollary 5.21 and by observing that $\forall Q \subseteq \mathscr{B}_{\Omega_1}$, $M(M'_1 \cup Q) \in \mathscr{H}(P_1)$. Then, since M'_1 is a compositional Ω_1-model of P_1, $M(M'_1 \cup Q)$ is a Herbrand model of $P_1 \cup Q$ and so $M(P_1 \cup Q) \subseteq M(M'_1 \cup Q)$. □

We show now that $\mathscr{O}_\Omega(P)$ is a compositional Ω-model (and hence an Ω-model). Note that $M(P) \subseteq \mathscr{O}_\Omega(P)$ since $M(P)$ is the least u-closed Ω-model and $\mathscr{O}_\Omega(P)$ is u-closed. Clearly, this ordering is strict, i.e. in general $\mathscr{O}_\Omega(P) \not\subseteq M(P)$. Consider for example the program $P = \{p(X), p(b)\}$. Then $\mathscr{O}_\emptyset(P) = P$ (considered as a denotation) and $P \neq M(P)$.

Theorem 5.23. *Let P be an Ω-open program. Then $\mathscr{O}_\Omega(P)$ is a compositional Ω-model of P.*

Proof (*by contradiction*). Assume that $\mathscr{O}_\Omega(P)$ is not a compositional Ω-model of P. Then $\exists J \in \mathscr{H}(\mathscr{O}_\Omega(P))$ such that J is not a Herbrand model of P. Then $\exists c = A :- B_1, \ldots, B_n \in P$ and $\exists \sigma$ such that $A\sigma$ is ground, $B_i\sigma \in J$, $\forall i = 1, \ldots, n$ and $A\sigma \neq J$. Since $J \in \mathscr{H}(\mathscr{O}_\Omega(P))$,

$$
\begin{aligned}
J &= M(\mathscr{O}_\Omega(P) \cup I) \\
&= I' \cup \{ D\vartheta \mid D\vartheta \text{ is ground, } D :- D_1, \ldots, D_k \in \mathscr{O}_\Omega(P) \cup Id_\Omega \\
&\qquad \text{and } D_i\vartheta \in I',\ i = 1, \ldots, k \qquad \},
\end{aligned}
$$

where $I \subseteq \mathscr{B}_\Omega$ and $I' = \{p(\tilde{t}) \mid p \in \Omega \text{ and } p(\tilde{t}) \in J\}$. Since $\forall i = 1, \ldots, n$, $B_i\sigma \in J$, there exists $B_i' :- D_{i,1}, \ldots, D_{i,k_i}$ variants of clauses in $\mathscr{O}_\Omega(P) \cup Id_\Omega$ and $\exists \sigma_i$ such that $B_i'\sigma_i = B_i\sigma$ and $D_{i,j}\sigma_i \in I'$, $\forall j = 1, \ldots, k_i$. Then, by definition of $\mathscr{O}_\Omega(P)$, $\exists (A :- D_{1,1}, \ldots, D_{1,k_1}, \ldots, D_{n,1}, \ldots, D_{n,k_n})\gamma \in \mathscr{O}_\Omega(P)$ and $\exists \sigma'$ such that $A\gamma\sigma' = A\sigma$ and $\forall i = 1, \ldots, n$, $\forall j = 1, \ldots, k_i$, $D_{i,j}\gamma\sigma' \in I'$. Thus, $A\gamma\sigma' = A\sigma \in J$ which contradicts the hypothesis. □

Corollary 5.24. *Let P be an Ω-open program. Then $\mathscr{O}_\Omega(P)$ is an Ω-model (according to Definition 5.3) of P.*

Proof. The proof is straightforward from Theorem 5.23, Definitions 5.3 and 5.18. □

In the following, we use the notation [ ] also for Ω-denotations, i.e. we denote by $[I]$ the Ω-denotation $\{c \in \mathscr{C}_\Omega \mid \exists c' \in I,\ c = \mu(c')\vartheta \text{ and } c \text{ is ground}\}$, where $\mu(c)$ gives one element of the equivalence class c. Clearly, the definition of $[I]$ is independent from the choice of the element $\mu(c')$.

We have shown that $\mathscr{O}_\Omega(P)$ is a compositional Ω-model. Clearly, since the $\approx_\Omega$ equivalence is finer than $\approx_{\Omega,g}$, the semantics $\mathscr{O}_\Omega(P)$ is correct w.r.t. $\approx_{\Omega,g}$, i.e. if $\mathscr{O}_\Omega(P_1) = \mathscr{O}_\Omega(P_2)$ then $P_1 \approx_{\Omega,g} P_2$. Obviously, since $\mathscr{O}_\Omega(P)$ also models computed answers, this semantics contains too much information to achieve also full abstractness w.r.t. $\approx_{\Omega,g}$. In order to achieve such a result, we then need an abstraction on $\mathscr{O}_\Omega(P)$.

First note that a goal G has a successful derivation iff a ground instance of G has a successful derivation. Therefore, for successful derivations it is sufficient to consider the ground version $[I]$ of the denotation I of a program. Note also that if a clause c subsumes c' then each successful derivation can be performed by using c instead of c'. Moreover, tautological clauses can be deleted. Indeed, since a tautology contains in the body, a copy of the head, if a successful derivation for the goal G in a program P uses a tautology c then there exists also a successful derivation for G which does not use c. All these observations allow to formally define the abstraction $\mathscr{M}_\Omega(P)$ of $\mathscr{O}_\Omega(P)$ as follows. We say that a clause c properly subsumes the clause c' iff c subsumes c' and c' does not subsume c.

Definition 5.25. Let Q be a set of ground clauses. Then we define

$$wcf(Q) = \{c \mid c = A :- B_1, \ldots, B_n \in \mathscr{C}_\Pi,\ B_i \neq B_j \text{ for } 1 \leqslant i \neq j \leqslant n$$
$$\exists A :- D_1, \ldots, D_m \in Q \text{ s.t. } \{B_1, \ldots, B_n\} = \{D_1, \ldots, D_m\}$$
$$c \text{ is not a tautology,}$$
$$\nexists c' \in Q \text{ such that } c' \text{ properly subsumes } c \quad \}.$$

Definition 5.26 (*Model-theoretic compositional semantics*). Let P be an Ω-open program. Then we define $\mathscr{M}_\Omega(P) = wcf([\mathscr{O}_\Omega(P)])$.

For a set Q of ground clauses, $wcf(Q)$ returns the clauses in Q, modified by deleting repetitions of atoms in the body, which are not subsumed by other clauses in Q, and which are not tautologies. Note that $wcf(Q)$ is the weak canonical form [46] of Q. The (weak) canonical form of a set P of (possibly nonground) clauses can be obtained [46] by "partitioning the clauses of P according to subsumption equivalence, by choosing the reduced clause for each equivalence class and then deleting those clauses which are subsumed by some other clauses (and tautologies)". A clause c is reduced [51] if it contains no identical atoms in the body and cannot be subsumed by a nonrenaming instance of c. Reference [46] shows that each clause is subsumption-equal (up to renaming) to a reduced clause and that the (weak) canonical form is unique up to renaming (considering bodies as sets). The following theorem shows the relation of these notions with the T_P operator. Recall that two sets of clauses P and Q are weakly subsumption equivalent if for every $c \in P$ which is not a tautology there exists $c' \in Q$ such that c' subsumes c and vice versa. Moreover in the following by $T_P + id = T_Q + id$ we mean that for every set of ground atoms $X, T_P(X) \cup X = T_Q(X) \cup X$ (id is the identity).

Theorem 5.27 (Maher [46]). *Let P, Q be sets of clauses. P and Q are weakly subsumption equivalent iff $T_P + id = T_Q + id$.*

Lemma 5.28. *Let P, Q be (possibly infinite) sets of ground clauses. Then P is weakly subsumption equivalent to Q iff $wcf(P) = wcf(Q)$.*

Proof. By definition of wcf, if $wcf(P)$ is subsumption equivalent to $wcf(Q)$ then $wcf(P) = wcf(Q)$. Then to prove the thesis we only need to show that $wcf(P)$ and P are weakly subsumption equivalent (w.s.e.). Since we are considering ground clauses, $c = A :- B_1, \ldots, B_n$ subsumes $c' = A :- D_1, \ldots, D_m$ iff $\{B_1, \ldots, B_n\} \subseteq \{D_1, \ldots, D_m\}$, and hence c and c' are subsumption equivalent iff $\{B_1, \ldots, B_n\} = \{D_1, \ldots, D_m\}$. Therefore, P and P' are subsumption equivalent, where P' is obtained from P by considering bodies of clauses as sets, and we have to show that $wcf(P)$ and P' are w.s.e. First note that, by definition of wcf, $wcf(P) = wcf(P') \subseteq P'$. Hence, we only have to show that $\forall c' \in P' \exists c \in wcf(P')$ such that c subsumes c'. Note that since we are considering ground clauses, for every $c' = A :- \{D_1, \ldots, D_m\}$ there exists only a finite number of different clauses $c_i = A :- \{B_1, \ldots, B_n\}$ (where bodies are considered as sets) such that $\{B_1, \ldots, B_n\} \subset \{D_1, \ldots, D_m\}$ (i.e. c_i subsumes c and not vice versa). Therefore, in P' there are no infinite chains $c_1 > c_2 > \cdots$, where $c_1 > c_2$ means c_1 is properly subsumed by c_2. Then, by definition of wcf, $\forall c' \in P'$, $\exists c \in wcf(P')$ such that c subsumes c' and the thesis holds. □

For finite sets, the nonground version of the previous lemma was proved in [46] (i.e. two finite sets of possibly nonground clauses are weakly subsumption equivalent iff they have the same weak canonical form). However, note that Lemma 5.28 does not

hold if we consider possibly nonground clauses, and therefore we consider wcf as the weak canonical form as in [46]. A counter example can be obtained by considering the program P given in [30]

$$P=\{c_1)\, A:\!-R(X_1,X_1)$$
$$c_2)\, A:\!-R(X_1,X_2),R(X_2,X_1)$$
$$\vdots$$
$$c_n)\, A:\!-R(X_1,X_2),R(X_2,X_3)\ldots R(X_{2^n-1},X_{2^n}),R(X_{2^n},X_1)$$
$$\vdots \qquad\qquad \},$$

since in this case we have an infinite chain $c_1 > c_2 > \cdots$ (c_1 properly subsumed by $c_2 \ldots$).

Theorem 5.31 shows the full abstraction result. A different fully abstract invariant w.r.t. $\approx_{\Omega,g}$ was already given in [30].

Lemma 5.29. *Let P_1, P_2 be programs. Then $P_1 \approx_{\Omega,g} P_2$ iff $[P_1] \approx_{\Omega,g} [P_2]$.*

Proof. Note that, in general, $M(P\cup Q)=M([P]\cup Q)$ (since $M(P\cup Q)=T_{P\cup Q}\uparrow\omega = T_{[P]\cup Q}\uparrow\omega = M([P]\cup Q)$). By Corollary 5.21, $P_1 \approx_{\Omega,g} P_2$ iff $\forall Q\subseteq \mathscr{B}_\Omega\ M(P_1\cup Q)= M(P_2\cup Q)$, and hence the thesis holds. □

Lemma 5.30. *Let Ω be a set of predicates and let P, W be u-closed sets of ground clauses such that if $H:\!-\tilde{B}\in P\cup W$ then $Pred(\tilde{B})\subseteq\Omega$. Then $P\approx_{\Omega,g} W$ iff $wcf(P)=wcf(W)$.*

Proof. By Lemma 5.28, since P and W are sets of ground clauses, $wcf(P)=wcf(W)$ iff P and W are weakly subsumption equivalent. By Theorem 5.27, P and W are weakly subsumption equivalent iff $T_P+id=T_W+id$. Moreover, by Corollary 5.21, $P\approx_{\Omega,g} W$ iff $\forall Q\subseteq\mathscr{B}_\Omega, M(P\cup Q)=M(W\cup Q)$. By hypothesis all the predicate symbols appearing in the bodies of clauses in P and W are contained in Ω. Then to prove the thesis we only need to prove that $\forall Q\subseteq\mathscr{B}_\Omega$, $M(P\cup Q)=M(W\cup Q)$ iff $T_P+id=T_W+id$. We prove the two implications separately.

"If": Since $T_{P\cup Q}(X)=T_P(X)\cup T_Q(X)$, if $T_P+id=T_W+id$ then $T_{P\cup Q}+id=T_{W\cup Q}+id$. Since $M(P)=T_P\uparrow\omega=T_P+id\uparrow\omega$, $M(P\cup Q)=M(W\cup Q)$.

"Only if": Let us consider a set Q of ground atoms. Note that, since P is u-closed, $M(P\cup Q)=T_{P\cup Q}(Q)=T_P(Q)\cup Q$. Analogously $M(W\cup Q)=T_W(Q)\cup Q$. Then, since $\forall Q, M(P\cup Q)=M(W\cup Q)$, $\forall Q$ set of ground atoms $T_P(Q)\cup Q=T_W(Q)\cup Q$, i.e. $T_P+id=T_W+id$ and this completes the proof. □

Theorem 5.31 (Full abstraction). *Let P_1, P_2 be Ω-open programs. Then, $\mathscr{M}_\Omega(P_1)=\mathscr{M}_\Omega(P_2)$ iff $P_1\approx_{\Omega,g} P_2$.*

Proof. First note that, by (a particular case of) Theorem 2.10, $P \approx_{\Omega,g} \mathcal{O}_\Omega(P)$ and hence $P_1 \approx_{\Omega,g} P_2$ iff $\mathcal{O}_\Omega(P_1) \approx_{\Omega,g} \mathcal{O}_\Omega(P_2)$. Therefore, we have the following implications:

$P_2 \approx_{\Omega,g} P_2$	iff (by the previous observation),
$\mathcal{O}_\Omega(P_1) \approx_{\Omega,g} \mathcal{O}_\Omega(P_2)$	iff (by Lemma 5.29),
$[\mathcal{O}_\Omega(P_1)] \approx_{\Omega,g} [\mathcal{O}_\Omega(P_2)]$	iff (by Lemma 5.30 since $[\mathcal{O}_\Omega(P)]$ is u-closed and ground),
$wcf([\mathcal{O}_\Omega(P_1)]) = wcf([\mathcal{O}_\Omega(P_2)])$	iff (by Definition 5.25),
$\mathcal{M}_\Omega(P_1) = \mathcal{M}_\Omega(P_2)$,	

and this completes the proof. □

Let us finally show that $\mathcal{M}_\Omega(P)$ is a compositional Ω-model. Then $\mathcal{M}_\Omega(P)$ can be considered as the compositional model-theoretic semantics of the program P.

Lemma 5.32. *Let I be an Ω-denotation. Then $\mathcal{H}(I) = \mathcal{H}(wcf([I]))$.*

Proof. Note that, by definition of Herbrand model, $M(I \cup J) = M([I] \cup J)$ and therefore $\mathcal{H}(I) = \mathcal{H}([I])$. Then to prove the thesis, we only need to show that $\mathcal{H}([I]) = \mathcal{H}(wcf([I]))$. Note that, by definition of wcf, $[I]$ and $wcf([I])$ are weakly subsumption equivalent (i.e. they have the same weak canonical form). Then, by Theorem 5.27, $T_{[I]} + id = T_{wcf([I])} + id$. Note that, in general, $T_{P \cup Q}(X) = T_P(X) \cup T_Q(X)$. Then, $\forall J$ set of atoms $T_{[I] \cup J} + id = T_{wcf([I]) \cup J} + id$, and therefore $M([I] \cup J) = M(wcf([I]) \cup J)$ (since $M(P) = T_P \uparrow \omega$). Then $\mathcal{H}([I]) = \mathcal{H}(wcf([I]))$, and the thesis holds. □

Proposition 5.33. *Let P be an Ω-open program. Then $\mathcal{M}_\Omega(P)$ is a compositional Ω-model.*

Proof. By Lemma 5.32, $\mathcal{H}(\mathcal{O}_\Omega(P)) = \mathcal{H}(wcf([\mathcal{O}_\Omega(P)]))$. By Theorem 5.23, $\mathcal{O}_\Omega(P)$ is a compositional Ω-model of P. Therefore, by Definition 5.18, $\mathcal{M}_\Omega(P) = wcf([\mathcal{O}_\Omega(P)])$ is a compositional Ω-model of P. □

6. S_Ω-models

We will now consider the relation between Ω-models (Definition 5.3) and the S_Ω-models defined in [7] on the same set of denotations. Both the Ω- and S_Ω-models are intended to capture specific operational properties, from a model-theoretic point of view. However, S_Ω-models are based on an ad hoc notion of truth (S_Ω-truth) and the

least S_Ω-model is exactly $\mathscr{F}_\Omega(P)$. Conversely, Ω-models are based on the usual notion of truth in a Herbrand interpretation through the function $\mathscr{H}'$. $\mathscr{F}_\Omega(P)$ is a nonminimal Ω-model.

Definition 6.1 (*Bossi and Menegus* [7]) (S_Ω-*truth*). Let Ω be a set of predicate symbols and I be an Ω-denotation. A definite clause $c = A :- B_1, \ldots, B_m$, $m \geq 0$ is S_Ω-true in I iff $unf_{\{c\}}(I \cup Id_\Omega) \subseteq I$.

S_Ω-models are defined in the obvious way. Note that, by definition of S_Ω-truth, $I \in \mathscr{D}$ is an S_Ω-model of P iff $unf_P(I \cup Id_\Omega) \subseteq I$.

Proposition 6.2. *Every S_Ω-model is an Ω-model (according to Definition* 5.3).

Proof. The proof is similar to that of Theorem 5.23. □

Proposition 6.3 (Bossi and Menegus [7]). *If Λ is a nonempty set of S_Ω-models of an Ω-open program P, then $\bigcap_{M \in \Lambda} M$ is an S_Ω-model of P.*

Proposition 6.3 allows to define the model-theoretic semantics $\mathscr{M}_{S_\Omega}(P)$ for a program P in terms of the S_Ω-models as follows.

Definition 6.4 (Bossi and Menegus [7]). *Let P be an Ω-open program and S be the set of all the S_Ω-models of P. Then $M_{S_\Omega}(P) = \bigcap_{M \in S} M$.*

Corollary 6.5. *Let Λ be a nonempty set of S_Ω-models of an Ω-open program P. Then $\bigcap_{M \in \Lambda} M$ is an Ω-model of P.*

By definition and by Proposition 6.3, $M_{S_\Omega}(P)$ is the least S_Ω-model in the lattice $(\mathscr{D}, \subseteq)$ (recall that $\mathscr{D}$ is the set of all the Ω-denotations). The following proposition shows that $M_{S_\Omega}(P)$ is also the least S_Ω-model in the lattice $(\mathscr{D}, \sqsubseteq)$.

Proposition 6.6. *Let P be a program and let S be the set of all the S_Ω-models of P. Then $M_{S_\Omega}(P) = glb(S)$ (according to the $\sqsubseteq$ ordering).*

Proof. By Lemma 6.3 $M_{S_\Omega}(P)$ is an S_Ω-model. Moreover, since $I \subseteq J$ implies $I \sqsubseteq J$, $\mathscr{M}_{S_\Omega}(P)$ is a lower bound (w.r.t. $\sqsubseteq$) of S. Therefore, $M_{S_\Omega}(P) = glb(S)$. □

The following theorem shows the equivalence of the least S_Ω-model $M_{S_\Omega}(P)$ and the fixpoint semantics (Definition 4.6).

Theorem 6.7 (Bossi and Menegus [7]). *Let P be an Ω-open program. Then $\mathscr{F}_\Omega(P) = M_{S_\Omega}(P)$.*

Corollary 6.8. *Let P be an Ω-open program. Then $\mathscr{F}_\Omega(P)$ is an Ω-model of P.*

Proof. The proof is straightforward from Theorem 6.7 and Proposition 6.2. □

It is worth noting that, since $\mathscr{O}_\Omega(P)=\mathscr{F}_\Omega(P)=M_{S_\Omega}(P)$, Theorem 2.13 shows that the least S_Ω-model $M_{S_\Omega}(P)$ is compositional w.r.t. Ω-union of programs when considering computed answer substitutions as observables. This result was already proved in [7] for the $M_{S_\Omega}(P)$ model. Finally note that, as shown by the following example, T_P^Ω is not monotonic (and therefore it is not continuous) on the complete lattice $(\mathscr{D}, \sqsubseteq)$. However, Proposition 6.10 ensures us that $\mathscr{F}_\Omega(P)$ is still the least fixpoint of T_P^Ω on $(\mathscr{D}, \sqsubseteq)$.

Example 6.9. Consider the program $P=\{p(x) \text{:-} q(x), r(b)\}$. Let $\Omega=\emptyset$, $I_1=\{q(a), q(x)\}$ and $I_2=\{r(b), q(x)\}$. Then $I_1 \sqsubseteq I_2$ while $T_P^\Omega(I_1)=\{p(x), p(a), r(b)\} \not\sqsubseteq T_P^\Omega(I_2)=\{p(x), r(b)\}$.

Proposition 6.10. *$T_P^\Omega \uparrow \omega$ is the least fixpoint of T_P^Ω on the complete lattice $(\mathscr{D}, \sqsubseteq)$.*

Proof. (a) *$\bigcup_{n\geq 0}(T_P^\Omega \uparrow n)$ is a fixpoint of T_P^Ω*: The proof is straightforward by Proposition 4.5.

(b) *$\bigcup_{n\geq 0} T_P^\Omega \uparrow n$ is the least fixpoint of T_P^Ω*: By continuity of T_P^Ω, $\bigcup_{n\geq 0} T_P^\Omega \uparrow n$ is the least fixpoint with respect to set inclusion. Then the thesis follows, since $I \subseteq J$ implies $I \sqsubseteq J$. □

7. Related work and applications

The result of our semantic construction has several similarities with the proof-theoretic semantics defined in [30, 31]. Our construction, however, is closer to the usual characterization of the semantics of logic programs. Namely, we define a top-down operational and a bottom-up fixpoint semantics and, last but not least, a model-theoretic semantics which allows us to obtain a declarative characterization of syntactically defined models. The semantics in [30] does not characterize computed answer substitutions, while the denotation defined by the fully abstract semantics in [31] is not a set of clauses (i.e. a program). The framework of [30, 31] can be useful for defining a program-equivalence notion, even if our more declarative (model-theoretic) characterization is even more adequate. Moreover, the presence of an operational or a fixpoint semantics makes our construction useful as a formal basis for program analysis.

Another related paper is [8], where Ω-open logic programs are called open theories. Open theories are provided with a model-theoretic semantics which is based on ideas very similar to those underlying our Definition 5.1. However, [8] does not consider semantic definitions in the style of our $\mathscr{O}_\Omega(P)$ which gives a unique denotation to any open program and does not consider computed answers as observables.

Our Ω-semantics $\mathcal{O}_\Omega(P)$ has already been used for several applications. Let us briefly discuss some of them.

- Suitable abstractions of the open semantics can be used to model nonstandard observables useful for program analysis. For example, in [28] a fully abstract semantics for partial answers is obtained essentially by considering the heads of the clauses in $\mathcal{O}_\Pi(P)$. Moreover, [28] defines also an extension of the Ω-semantics which takes into account the selection rule and which allows to correctly model those observables which depend on it. $\mathcal{O}_\Pi(P)$ is also useful for studying new equivalences of logic programs [26, 25] based on computed answer substitutions which are not considered in [46].
- The open semantics is useful to model incomplete knowledge bases, where new chunks of knowledge can incrementally be assimilated, and logic languages provided with a module-like structure. Modified versions of $\mathcal{O}_\Omega(P)$ allow to obtain semantics compositional w.r.t. various composition operators. A semantic compositional w.r.t. a generalized inheritance operator is obtained in [4]. Static and dynamic extension/overriding mechanisms can be expressed using the generalized operator. Since the semantic domains are (equivalence classes of) clauses, we have a uniform treatment of static and dynamic inheritance which in other compositional semantics [9] require different semantic objects to coexist. Moreover, the semantics in [4] is the first compositional semantics of units and inheritance which correctly models computed answer substitutions.
- The open semantics can be considered as the semantic base for modular program analysis. It may happen that in a system under development, not all the pieces of the program are available for analysis at some point, but we might want to do some analysis in any case. Clearly, this is possible only if the base semantics is modular, i.e. compositional. Modularity helps considerably in reducing the complexity of analysis and in proving correctness of programs, since it allows an incremental and structured specification and verification of the software. A first use of our semantics for modular analysis is in [12].

Let us finally remark some other interesting properties of the Ω-semantics $\mathcal{O}_\Omega(P)$ which could be further investigated.

- By means of a syntactic device, we obtain a unique representation for a possibly infinite set of Herbrand models when a unique representative Herbrand model does not exist. A similar device was used in [15, 35, 27] to characterize logic programs with negation. A combination of these results with our semantics could be considered to develop a compositional semantics for programs with negation.
- Our framework is strongly related to *abduction* [17]. If Ω is the set of abducible predicates, the abductive consequences of any goal G can be found by executing G in $\mathcal{O}_\Omega(P)$.
- The delayed evaluation of open predicates which is typical of $\mathcal{O}_\Omega(P)$ can easily be generalized to other logic languages, to achieve compositionality w.r.t. the union of programs. In particular, this matches quite naturally the semantics of CLP and concurrent constraint programs given in [22].

- The modification of the $\mathcal{O}_\Omega(P)$ semantics developed in [4] could be used to develop a modular analysis for programs which are structured by using inheritance mechanisms, according to the usual object-oriented style.

Appendix

In this appendix we give the proofs of some lemmata and theorems. Let us first introduce some definitions about substitutions and equations. The interested reader can see [16, 40, 50]. An equation is an atom $s=t$, where s,t are terms and $=$ is a predicate symbol which is interpreted as the syntactic equality over the Herbrand universe. Given a set of equations $E=\{s_1=t_1,\ldots,s_n=t_n\}$, the (most general) unifier of (E) is a (most general) unifier of $((s_1,\ldots,s_n)(t_1,\ldots,t_n))$. It is well known that the idempotent mgu of a set of equations (terms) is unique up to renaming (see e.g. [16]). Moreover, if a set of equations E is unifiable, then there exists an idempotent mgu of E. A solution of E is a grounding unifier of E. Two sets of equations E_1, E_2 are equivalent ($E_1 \approx_e E_2$) if they have the same solutions. The lattice structure on idempotent substitutions [16] is isomorphic to the lattice structure on equations introduced in [40] ([50] extends this result to include also the trivial Herbrand universe). Therefore, we can use indifferently equations or idempotent mgu's. In the following we will always implicitly consider idempotent mgu's and a nontrivial Herbrand universe. Moreover, idempotent mgu's are considered equal up to renaming, i.e. if $\vartheta_1\rho=\vartheta_2$, where ρ is a renaming, we write (by a slight abuse of notation) $\vartheta_1=\vartheta_2$. $\tilde{t}=\tilde{s}$ denotes a set of equations.

Definition A.1. Let $\vartheta=\{X_1/t_1,\ldots,X_n/t_n\}$ be a substitution. Then $\mathcal{E}(\vartheta)=\{X_1=t_1,\ldots,X_n=t_n\}$.

Lemma A.2 (Palamidessi [50]). *Let ϑ_1,ϑ_2 be idempotent substitutions. Then $mgu(\mathcal{E}(\vartheta_1)\cup\mathcal{E}(\vartheta_2))=\vartheta_1\, mgu(\mathcal{E}(\vartheta_2)\vartheta_1)=\vartheta_2\, mgu(\mathcal{E}(\vartheta_1)\vartheta_2)$.*

Theorem A.3 (Lassez [40]). *If the Herbrand universe is non trivial, then ϑ is an idempotent mgu of E iff $\mathcal{E}(\vartheta)\approx_e E$.*

If ϑ is an idempotent mgu of E, $\mathcal{E}(\vartheta)$ is called the solved form of E [40].

Corollary A.4. *If the Herbrand universe is nontrivial and $E_1\approx_e E_2$, then for any set of equations E, $mgu(E\cup E_1)=mgu(E\cup E_2)$.*

Proof. By definition of solution, if $E_1\approx_e E_2$ then $E\cup E_1\approx_e E\cup E_2$. Then the thesis follows from Theorem A.3.

Corollary A.5. *Let E_1,E_2 be sets of equations with $mgu(E_1)=\vartheta$ and $mgu(E_2\vartheta)=\gamma$. Then $mgu(E_1\cup E_2)=\vartheta\gamma$.*

Proof. We have the following equalities

$$\begin{aligned} mgu(E_1 \cup E_2) &= mgu(\mathscr{E}(\vartheta) \cup E_2) && \text{(by Corollary A.4, and by Theorem A.3)} \\ &= \vartheta mgu(E_2 \vartheta) && \text{(by Lemma A.2)} \\ &= \vartheta\gamma && \text{(by Definition of } \gamma) \end{aligned}$$

and this completes the proof. □

In order to prove the following results we will consider a CLP-like [33] version of SLD derivation, denoted by $\rightsquigarrow^*_{P,R}$, which uses equations instead of mgu's. The previous stated isomorphism allows to prove the equivalence of the two versions of SLD (see below). More precisely, given a goal $p(\tilde{s}), G, E$, where G is a set of atoms, E is a set of equations and $p(\tilde{s})$ is the selected atom, and given a clause $c = p(\tilde{t}) :- B_1, \ldots, B_n \in P$, we can define $p(\tilde{s}), G, E \rightsquigarrow^*_{P,R} \tilde{s} = \tilde{t}, E, B_1, \ldots, B_n, G$ in one step (using clause c) iff the set $E \cup \{\tilde{s} = \tilde{t}\}$ is unifiable. Provided that no equational atom is considered (and hence chosen) by the selection rule and by replacing the atom $p(\tilde{t})$ by $p(\tilde{X})$, $\tilde{X} = \tilde{t}$ (where $\tilde{X}$ are new distinct variables), we can then replace a derivation $d = (p(\tilde{t}) \overset{\vartheta_2}{\rightsquigarrow}_{P,R} G')$ by the equivalent derivation $p(\tilde{X}), \tilde{X} = \tilde{t} \rightsquigarrow^*_{P,R} G, E$, which uses the same clauses as d. The equivalence is formally proved by the following result.

Lemma A.6 (Wolfram et al. [56]). *Let P be a program, let $\rightsquigarrow^*$ be defined as before and let $G = A_1, \ldots, A_k$ be a goal, where $A_i = p_i(\tilde{t}_i)$, $i = 1, \ldots, k$. Let $H = p_1(\tilde{X}_1), \ldots, p_k(\tilde{X}_k)$, $E = \{\tilde{X}_1 = \tilde{t}_k \cup \cdots \cup \tilde{X}_k = \tilde{t}_k\}$ and let $\beta = \{x/t \mid x = t \in \tilde{X}_i = \tilde{t}_i,\ 1 \leqslant i \leqslant k\}$, with $\tilde{X}_1, \ldots, \tilde{X}_k$ new distinct variables. Then $G \overset{\vartheta}{\rightsquigarrow}_{P,R} G'$ iff $H, E \rightsquigarrow^*_{P,R} H', E'$, with $\beta\vartheta = mgu(E')$ and $G' = H'\beta\vartheta$. Moreover, $mgu(E')|_{Var(G)} = \vartheta|_{Var(G)}$.*

Note that the previous lemma holds for any selection rule. Then we have the following lemma.

Lemma A.7 (Lemma 2.12). *Let P be a program and let G be a goal. Then $G \overset{\vartheta}{\rightsquigarrow}_{P,R} N$ iff $G \overset{\vartheta}{\rightsquigarrow}_{P,R'} N$, where $\vartheta|_{Var(G)} = \vartheta'|_{Var(G)}$ and the derivation $G \overset{\vartheta'}{\rightsquigarrow}_{P,R'} N$ is obtained from $G \overset{\vartheta}{\rightsquigarrow}_{P,R} N$ by changing the order in which the atoms are selected.*

Proof. Straightforward from Lemma A.6, by noting that in $\rightsquigarrow^*_{P,R}$ derivations the computation is performed by accumulating equations and, since equations are considered as sets, the ordering in which equations are added is not relevant. □

Proposition A.8 (Proposition 2.6). *Let R be a given fair selection rule, let $P^* = P \cup Id_\Omega$, $\tilde{X}$ be new distinct variables and $Pred(B_1, \ldots, B_n) \subseteq \Omega$. Then there exists a rule R' such that $p(\tilde{X}) \overset{\vartheta}{\rightsquigarrow}_{P,R'} B_1, \ldots, B_n$ iff $p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P,R} D_1, \ldots, D_m \overset{\sigma}{\rightsquigarrow}_{P^*,R} B_1, \ldots, B_n$ and $p(\tilde{X})\gamma\sigma = p(\tilde{X})\vartheta$.*

Proof. ($\Leftarrow$) The proof is straightforward by considering R' as the selection rule obtained from R, by eliminating in the derivation,

$$p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P,R} D_1, \ldots, D_m \overset{\sigma}{\rightsquigarrow}_{P^*,R} B_1, \ldots, B_n$$

all the selections of atoms which are rewritten by clauses in $Id_\Omega \backslash P$.

($\Rightarrow$) In order to prove the thesis, we consider $\rightsquigarrow^*$ derivations. By hypothesis and by Lemma A.6, there exists a derivation $d_1 = p(\tilde{X}) \rightsquigarrow^*_{P,R'} E, B'_1, \ldots, B'_n$ such that E is a set of equations, $\vartheta = mgu(E)$ and $B_1, \ldots, B_n = (B'_1, \ldots, B'_n)\vartheta$. Then we can define a derivation $d_2 = p(\tilde{X}) \rightsquigarrow^*_{P^*,R} F, B'_1, \ldots, B'_n$ as follows. The first clause used in d_2 is the same clause of program P which is used to rewrite $p(\tilde{X})$ in the first step of the derivation d_1. For each atom A_j that is selected by R in a step of the derivation d_2, if A_j is selected by R' in d_1, then we use the same input clause, used in d_1 (recall that $P \subseteq P^*$). Note that if A_j is not selected by R', then $Pred(A_j) \in \Omega$, since $Pred(B_1, \ldots, B_n) \subseteq \Omega$. Therefore, if A_j is not selected by R', we can use, to rewrite A_j, the input clause $p_j(\tilde{X}_j) :- p_j(\tilde{X}_j) \in Id_\Omega$, where $p_j = Pred(A_j)$. Then the derivation d_2 uses only clauses which are used in d_1 and some clauses in Id_Ω. Moreover, since the selection rule R is fair, if an atom A_i is selected in a step of the derivation d_1, then in the derivation d_2 the atom A_i is selected within a finite number of steps (recall that we can always rewrite an atom $q(\tilde{t})$, where $q \in \Omega$ in the derivation d_2 by means of the input clause $q(\tilde{Y}) :- q(\tilde{Y})$). Thus, $F = E \cup E'$, where E' is a set of the equation $\{\tilde{t}_1 = \tilde{Y}_1, \ldots, \tilde{t}_s = \tilde{Y}_s\}$, where, for $i = 1, \ldots, s$, $\tilde{Y}_i$ are new distinct variables, which do not occur in E. Then it is easy to check that $\exists \sigma = mgu(F)$, $\sigma = \beta\vartheta$, where $\beta = (\tilde{Y}_1/\tilde{t}_1, \ldots, \tilde{Y}_s/\tilde{t}_s)$. Therefore, by Lemma A.6,

$$p(\tilde{X}) \overset{\sigma}{\rightsquigarrow}_{P^*,R} (B'_1, \ldots, B'_n)\sigma,$$

where $B_1, \ldots, B_n = (B'_1, \ldots, B'_n)\sigma$ and $p(\tilde{X})\sigma = p(\tilde{X})\rho\vartheta = p(\tilde{X})\vartheta$, and this concludes the proof. □

Lemma A.9 (Lemma 2.9). *Let P be a program. Then $p(\tilde{t}) \overset{\vartheta_1}{\rightsquigarrow}_{P,R} G_1$ iff the following hold*

- $p(\tilde{X}) \overset{\vartheta_2}{\rightsquigarrow}_{P,R} G_2$,
- *there exists $\sigma = mgu(p(\tilde{t}), p(\tilde{X})\vartheta_2)$, $p(\tilde{t})\vartheta_1 = p(\tilde{X})\vartheta_2\sigma$ and $G_2\sigma = G_1$.*

Proof. Let us define $\beta = \{\tilde{X}/\tilde{t}\}$. By Lemma A.6

$$p(\tilde{X}) \overset{\vartheta_2}{\rightsquigarrow}_{P,R} G_2$$

iff

$$p(\tilde{X}) \rightsquigarrow^*_{P,R} E, G$$

where $\vartheta_2 = mgu(E)$, $G\vartheta_2 = G_2$ and

$$p(\tilde{t}) \overset{\vartheta_1}{\rightsquigarrow}_{P,R} G_1$$

iff

$$p(\tilde{X}), \tilde{X}=\tilde{t} \rightsquigarrow^{*}_{P,R} G, E, \tilde{X}=\tilde{t}$$

where $\beta\vartheta_1=\gamma=mgu(E\cup\{\tilde{X}=\tilde{t}\})$ and $G\gamma=G_1$. By definition of the unification algorithm and by definition of mgu, $E\cup\{\tilde{X}=\tilde{t}\}$ is unifiable iff $\tilde{t}=\tilde{X}\vartheta_2$ is unifiable iff there exists $\sigma=mgu(p(\tilde{t}), p(\tilde{X})\vartheta_2)$. Therefore, by the previous equivalence of derivations $\rightsquigarrow^{*}$ and $\rightsquigarrow$,

$$p(\tilde{X}) \overset{\vartheta_2}{\rightsquigarrow}_{P,R} G_2,$$

iff

$$p(\tilde{t}) \overset{\vartheta_1}{\rightsquigarrow}_{P,R} G_1$$

and there exists $\sigma=mgu(p(\tilde{t}), p(\tilde{X})\vartheta_2)$. By definition of $\beta, \vartheta_1, \gamma$, $p(\tilde{t})\vartheta_1=p(\tilde{X})\beta\vartheta_1=p(\tilde{X})\gamma$. Now

$$\begin{aligned}
\vartheta_2\sigma &= \vartheta_2 mgu(\{\tilde{X}\vartheta_2=\tilde{t}\}) && \text{(by definition of } \sigma)\\
&= \vartheta_2 mgu((\{\tilde{X}=\tilde{t}\})\vartheta_2) && \text{(by definition of } \vartheta_2)\\
&= mgu(E\cup\{\tilde{X}=\tilde{t}\}) && \text{(by Corollary A.5)}\\
&= \gamma && \text{(by definition of } \gamma).
\end{aligned}$$

Therefore, $p(\tilde{X})\vartheta_2\sigma=p(\tilde{X})\gamma=p(\tilde{t})\vartheta_1$. Moreover, by definition of G_1, G_2 and $\gamma, G_1=G\gamma=G\vartheta_2\sigma=G_2\sigma$ and this completes the proof. □

In the proof of the following theorem we use a parallel derivation and its equivalence to the SLD derivation with a fair rule. Note that the equivalence of the fair SLD derivations and the parallel derivation was already proved in [56]. In the proof we show a simple argument for such an equivalence for the sake of completeness.

Theorem A.10 (Theorem 4.14). *Let P be an Ω-open program. Then $\mathcal{O}_\Omega(P)=\mathcal{U}_\Omega(P)$.*

Proof. Let us denote by $A_1, \ldots, A_m \overset{\vartheta}{\rightsquigarrow}_{P,\|} B_1, \ldots, B_n$ a parallel derivation step where the m atoms $A_1, \ldots, A_m$ are rewritten by the m clauses $H_i :- \tilde{B}_i \in P$ such that $\vartheta=mgu((A_1, \ldots, A_m)\ (H_1, \ldots, H_m))$ and $(\tilde{B}_1, \ldots, \tilde{B}_m)\vartheta=B_1, \ldots, B_n$. Let us define $P^*=P\cup Id_\Omega$. By Definition 4.9 (and by a straightforward inductive argument) $p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P,\|} A_1, \ldots, A_m \overset{\vartheta_2}{\rightsquigarrow}_{P^*,\|} B_1, \ldots, B_n$ in k steps iff $p(\tilde{X})\gamma\vartheta :- B_1, \ldots, B_n \in P_k$. By definition of $\mathcal{O}_\Omega(P)$ and $\mathcal{U}_\Omega(P)$, to prove the thesis it is then sufficient to show that, for $Pred(B_1, \ldots, B_n)\subseteq\Omega$,

$$p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P,\|} A_1, \ldots, A_m \overset{\vartheta}{\rightsquigarrow}_{P^*,\|} B_1, \ldots, B_n \quad \text{iff}$$

$$p(\tilde{X}) \overset{\gamma}{\rightsquigarrow}_{P,R} A_1, \ldots, A_m \overset{\vartheta}{\rightsquigarrow}_{P^*,R} B_1, \ldots, B_n$$

for a fair selection rule R. By Proposition 2.6, we can assume that R is a fair rule such that if it selects an atom A_j in the resolvent $G=A_1,\ldots,A_m$, all the atoms derived by A_j are evaluated (i.e. selected) after the atoms A_i for $i=1,\ldots,m$, $i\neq j$. Formally, we can define such a rule R_ϕ as follows.

Let ϕ be a given bijection on the set of natural numbers. Then R_ϕ selects in every resolvent the atom A with the minimum value $\psi(A)$, where the function $\psi(A)$ is defined inductively on the length h of derivation as follows.

For $h=0$: If $G_0=A_1,\ldots,A_m$ is the goal then $\psi(A_j)=\phi(j)$ for $1\leqslant j\leqslant n$.

For $h>0$: Assume a given value $\psi(A_j)$ for the atoms in $G_h=A_1,\ldots,A_m$ and let $G_{h+1}=(A_1,\ldots,A_{i-1},B_1,\ldots,B_r,A_{i+1},\ldots,A_m)\vartheta$ be the resolvent obtained by replacing $A_i=R_\phi(G_{h+1})$ by $B_1,\ldots,B_r$. Then for $1\leqslant k\leqslant r$, $\psi(B_k)=\phi(k)+max_{1\leqslant j\leqslant n}(\psi(A_j))$.

It is clear from the definition that R_ϕ is a fair rule. To simplify the notation we assume in the following that the function ϕ selects atoms from left to right. Note that

$$p(\tilde{X})\overset{\gamma}{\rightsquigarrow}_{P,R}A_1,\ldots,A_m$$

in one step iff $c=p(\tilde{X})\gamma:\!-A_1,\ldots,A_m\in P$. Moreover, observe that if

$$A_1,\ldots,A_m\overset{\vartheta}{\rightsquigarrow}_{P^*,R_\phi}B_1,\ldots,B_n$$

and the length of such a derivation is m, then by definition of R_ϕ for $i=1,\ldots,m$ each atom A_i is selected in such a derivation and rewritten by a clause in P^*.

Therefore, by (a slight modification of) Lemma 2.12,

$$p(\tilde{X})\overset{\gamma}{\rightsquigarrow}_{P,R_\phi}A_1,\ldots,A_m\overset{\vartheta}{\rightsquigarrow}_{P^*,R_\phi}B_1,\ldots,B_n \quad \text{in } m+1 \text{ steps iff}$$

$$p(\tilde{X})\overset{\gamma}{\rightsquigarrow}_{P,\|}A_1,\ldots,A_m\overset{\vartheta}{\rightsquigarrow}_{P^*,\|}B_1,\ldots,B_n \quad \text{in two steps}$$

Observe also that if

$$p(\tilde{X})\overset{\gamma}{\rightsquigarrow}_{P,R_\phi}A_1,\ldots,A_m\overset{\vartheta}{\rightsquigarrow}_{P^*,R_\phi}B_1,\ldots,B_r,A_k,\ldots,A_m$$

in $k<m+1$ steps and $Pred(B_1,\ldots,B_r,A_k,\ldots,A_m)\subseteq\Omega$ then we can obtain an equivalent derivation whose length is $m+1$ steps by simply rewriting the atoms $A_k,\ldots,A_m$ by clauses in Id_Ω. Therefore, by a straightforward inductive argument, we can prove that, for $Pred(B_1,\ldots,B_r,A_k,\ldots,A_m)\subseteq\Omega$,

$$p(\tilde{X})\overset{\gamma}{\rightsquigarrow}_{P,\|}A_1,\ldots,A_m\overset{\vartheta}{\rightsquigarrow}_{P^*,\|}B_1,\ldots,B_n \quad \text{iff}$$

$$p(\tilde{X})\overset{\gamma}{\rightsquigarrow}_{P,R_\phi}A_1,\ldots,A_m\overset{\vartheta}{\rightsquigarrow}_{P^*,R_\phi}B_1,\ldots,B_n$$

and this completes the proof. □

References

[1] K.R. Apt, Introduction to logic programming, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics* (Elsevier, Amsterdam, 1990) 493–574.

[2] R. Barbuti, M. Codish, R. Giacobazzi and G. Levi, Modelling prolog control, in: *Proc. 19th Ann. ACM Symp. on Principles of Programming Languages* (1992) 95–104.

[3] R. Barbuti, R. Giacobazzi and G. Levi, A general framework for semantics-based bottom-up abstract interpretation of logic programs, *ACM Trans. Programming Languages Systems* **15** (1993) 133–181.

[4] A. Bossi, M. Bugliesi, M. Gabbrielli, G. Levi and M.C. Meo, Differential logic programming, in: *Proc. 20th Ann. ACM Symp. on Principles of Programming Languages* (1993) 359–370.

[5] A. Bossi and N. Cocco, Basic transformation operations which preserve computed answer substitutions of logic programs, *J. Logic Programming* **16** (1993) 47–87.

[6] A. Bossi, M. Gabbrielli, G. Levi and M.C. Meo, Contributions to the semantics of open logic programs, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems 1992* (1992) 570–580.

[7] A. Bossi and M. Menegus, Una semantica composizionale per programmi logici aperti, in: P. Asirelli, ed., *Proc. 6th Italian Conf. on Logic Programming* (1991) 95–109.

[8] A. Brogi, E. Lamma and P. Mello, Composing open logic programs, *J. Logic and Computation*, to appear.

[9] M. Bugliesi, A declarative view of inheritance in logic programming, in: K. Apt, ed., *Proc. Joint Internat. Conf. and Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1992) 113–130.

[10] K.L. Clark, Predicate logic as a computational formalism, Research Report DOC 79/59, Imperial College, Dept. of Computing, London, 1979.

[11] M. Codish, D. Dams and E. Yardeni, Bottom-up abstract interpretation of logic programs, Technical Report, Dept. of Computer Science, The Weizmann Institute, Rehovot, 1990; *Theoret. Comput. Sci.* **124**, to appear.

[12] M. Codish, S.K. Debray and R. Giacobazzi, Compositional analysis of modular logic programs, in: *Proc. 20th Ann. ACM Symp. on Principles of Programming Languages* (1993) 451–464.

[13] F.S. de Boer and C. Palamidessi, Concurrent logic languages: asynchronism and language comparison, in: S. Debray and M. Hermenegildo, eds., *Proc. North American Conf. on Logic Programming '90*, (MIT Press, Cambridge, MA, 1990) 99–114.

[14] F. Denis and J.-P. Delahaye, Unfolding, procedural and fixpoint semantics of logic programs, in: C. Choffrut and M. Jantzen, eds., *STACS 91*, Lecture Notes in Computer Science, Vol. 480 (Springer, Berlin, 1991) 511–522.

[15] P.M. Dung and K. Kanchanasut, A fixpoint approach to declarative semantics of logic programs, in: E. Lusk and R. Overbeck, eds., *Proc. North American Conf. on Logic Programming '89* (MIT Press, Cambridge, MA, 1989) 604–625.

[16] E. Eder, Properties of substitutions and unifications, *J. Symbolic Comput.* **1** (1985) 31–46.

[17] K. Eshghi and R.A. Kowalski, Abduction compared with negation by failure, in: G. Levi and M. Martelli, eds., *Proc. 6th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1989) 234–254.

[18] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, A new declarative semantics for logic languages, in: R.A. Kowalski and K.A. Bowen, eds., *Proc. 5th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1988) 993–1005.

[19] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, Declarative modeling of the operational behavior of logic languages, *Theoret. Comput. Sci.* **69** (1989) 289–318.

[20] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, A model-theoretic reconstruction of the operational semantics of logic programs, Technical Report TR 32/89, Dipartimento di Informatica, Università di Pisa, 1989; *Inform. and Comput.*, to appear.

[21] G. Ferrand, Error diagnosis in logic programming, an adaptation of E.Y. Shapiro's method, *J. Logic Programming* **4** (1987) 177–198.

[22] M. Gabbrielli and G. Levi, Modeling answer constraints in constraint logic programs, in: K. Furukawa, ed., *Proc. Eighth Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1991) 238–252.

[23] M. Gabbrielli and G. Levi, On the semantics of logic programs, in: J. Leach Albert, B. Monien and M. Rodriguez-Artalejo, eds., *Automata, Languages and Programming, 18th Internat. Colloquium*, Lecture Notes in Computer Science, Vol. 510 (Springer, Berlin, 1991) 1–19.

[24] M. Gabbrielli and G. Levi, Unfolding and Fixpoint Semantics of Concurrent Constraint Programs, *Theoret. Comput. Sci.* **105** (1992) 85–128.

[25] M. Gabbrielli, G. Levi and M.C. Meo, Observable behaviors and equivalences of logic programs, Research Report IIAS-RR-92-9E, International Institute for Advanced Study of Social Information Science, FUJITSU LAB. Ltd., 1992.

[26] M. Gabbrielli, G. Levi and M.C. Meo, Observational equivalences for logic programs, in: K. Apt, ed., *Proc. Joint Internat. Conf. Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1992) 131–145.

[27] M. Gabbrielli, G. Levi and D. Turi, A two steps semantics for logic programs with negation, in: *Proc. Internat. Conf. on Logic Programming and Automated Reasoning*, Lecture Notes in Artificial Intelligence, Vol. 624 (Springer, Berlin, 1992) 297–308.

[28] M. Gabbrielli and M.C. Meo, Fixpoint semantics for partial computed answer substitutions and call patterns, in: H. Kirchner and G. Levi, eds., *Proc. 3rd Internat. Conf. on Algebraic and Logic Programming*, Lecture Notes in Computer Science, Vol. 632 (Springer, Berlin, 1992) 84–99.

[29] H. Gaifman, M.J. Maher and E.Y. Shapiro, Reactive behavior semantics for concurrent constraint logic programs, in: E. Lusk and R. Overbeck, eds., *Proc. North American Conf. on Logic Programming '89* (MIT Press, Cambridge, MA, 1989) 553–572.

[30] H. Gaifman and E. Shapiro, Fully abstract compositional semantics for logic programs, in: *Proc. 16th Ann. ACM Symp. on Principles of Programming Languages* (1989) 134–142.

[31] H. Gaifman and E. Shapiro, Proof theory and semantics of logic programs, in: *Proc. 4th IEEE Symp. on Logic In Computer Science* (IEEE Computer Society Press, Silver Spring, MD, 1989) 50–62.

[32] R. Giacobazzi, S.K. Debray and G. Levi, A generalized semantics for constraint logic programs, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems 1992* (1992) 581–591.

[33] J. Jaffar and J.-L. Lassez, Constraint logic programming, in *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages* (1987) 111–119.

[34] J. Jaffar and J.-L. Lassez, Constraint logic programming, Technical Report, Department of Computer Science, Monash University, June 1986.

[35] K. Kanchanasut and P. Stuckey, Eliminating negation from normal logic programs, in: H. Kirchner and W. Wechler, eds., *Proc. 2nd Internat. Conf. on Algebraic and Logic Programming*, Lecture Notes in Computer Science, Vol. 463 (Springer, Berlin, 1990) 217–231.

[36] T. Kawamura and T. Kanamori, Preservation of stronger equivalence in unfold/fold logic programming transformation, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems* (Institute for New Generation Computer Technology, Tokyo, 1988) 413–422.

[37] R. Kemp and G. Ringwood, An algebraic framework for the abstract interpretation of logic programs, in: S. Debray and M. Hermenegildo, eds., *Proc. North American Conf. on Logic Programming '90* (MIT Press, Cambridge, MA, 1990) 506–520.

[38] G. Kreisel and J.L. Krivine, *Elements of Mathematical Logic (Model Theory)* (North-Holland, Amsterdam, 1967).

[39] J.-L. Lassez and M.J. Maher, Closures and fairness in the semantics of programming logic, *Theoret. Comput. Sci.* **29** (1984) 167–184.

[40] J.-L. Lassez, M.J. Maher and K. Marriott, Unification revisited, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988) 587–625.

[41] G. Levi, Models, Unfolding rules and fixpoint semantics, in: R.A. Kowalski and K.A. Bowen, eds., *Proc. 5th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1988) 1649–1665.

[42] G. Levi and P. Mancarella, The unfolding semantics of logic programs, Technical Report TR-13/88, Dipartimento di Informatica, Università di Pisa, 1988.

[43] G. Levi and C. Palamidessi, An approach to the declarative semantics of synchronization in logic languages, in: J.-L. Lassez, ed., *Proc. 4th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1987) 877–893.

[44] J.W. Lloyd, *Foundations of Logic Programming* (Springer, Berlin, 2nd ed., 1987).

[45] J.W. Lloyd and J.C. Shepherdson, Partial evaluation in logic programming. *J. Logic Programming* **11** (1991) 217–242.

[46] M.J. Maher, Equivalences of logic programs, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988) 627–658.

[47] P. Mancarella and D. Pedreschi, An algebra of logic programs, in: R.A. Kowalski and K.A. Bowen, eds., *Proc. 5th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1988) 1006–1023.

[48] K. Marriott and H. Søndergaard, Semantics-based dataflow analysis of logic programs, in: G. Ritter, ed., *Information Processing 89* (North-Holland, Amsterdam, 1989) 601–606.

[49] R.A. O'Keefe, Towards an algebra for constructing logic programs, in: *Proc. IEEE Symp. on Logic Programming* (1985) 152–160.

[50] C. Palamidessi, Algebraic properties of idempotent substitutions, in: M.S. Paterson, ed., *Proc. 17th Internat. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 443 (Springer, Berlin, 1990) 386–399.

[51] G. Plotkin, A note on inductive generalization, *Machine Intelligence*, **5** (1970) 153–165.

[52] H. Rasiowa and R. Sikorski, *The Mathematics of Metamathematics* (North-Holland, Amsterdam, 1963).

[53] V.A. Saraswat and M. Rinard, Concurrent constraint programming, in: *Proc. 17th ACM Symposium on Principles of Programming Languages* (1990) 232–245.

[54] D. Turi, Extending S-models to logic programs with negation, in: K. Furukawa, ed., *Proc. 8th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1991) 397–411.

[55] M.H. van Emden and R.A. Kowalski, The semantics of predicate logic as a programming language, *J. ACM* **23** (1976) 733–742.

[56] D.A. Wolfram, M.J. Maher and J-L. Lassez, A unified treatment of resolution strategies for logic programs, in: Sten-Åke Tärnlund, ed., *Proc. 2nd Internat. Conf. on Logic Programming* (1984) 263–276.

[57] S. Yamasaki, M. Yoshida and S. Doshita, A fixpoint semantics of Horn sentences based on substitution sets, *Theoret. Comput. Sci.* **51** (1987) 309–324.